# Historic SE: Insights from Deluxe Paint for the Amiga

Giuseppe Destefanis,
Yann-Gaël Guéhéneuc, and Fabio Calefato

ReAnimate'26

26/06/12

Université Concordia University
GINA CODY SCHOOL OF ENGINEERING AND COMPUTER SCIENCE
Department of Computer Science & Software Engineering

https://www.moma.org/collection/works/61240


https://www.reddit.com/r/vintagecgi/comments/1g6l0cr/andy_warhol_messing_around_with_propaint_at_the/


Color
-- Deluxe Paint --
by Daniel Silva
(c) 1986 by Electronic Arts
UNDO
CLR

# COMPUTER HISTORY MUSEUM SOFTWARE LICENSE AGREEMENT – DELUXEPAINT SOURCE CODE [163.17KB]

PLEASE READ THE FOLLOWING TERMS AND CONDITIONS CAREFULLY BEFORE DOWNLOADING, INSTALLING OR USING THIS SOFTWARE (THE "SOFTWARE"). THE TERMS AND CONDITIONS OF THIS SOFTWARE LICENSE AGREEMENT ("AGREEMENT") GOVERN USE OF THE SOFTWARE.

Image: https://en.wikipedia.org/wiki/File:Amiga_1000DP.jpg
Icons: Freepik at Flaticon, e.g., https://www.flaticon.com/free-icon/ram_543279

Image: https://en.wikipedia.org/wiki/File:Amiga_1000DP.jpg
Icons: Freepik at Flaticon, e.g., https://www.flaticon.com/free-icon/ram_543279

Image: https://en.wikipedia.org/wiki/File:Amiga_1000DP.jpg
Icons: Freepik at Flaticon, e.g., https://www.flaticon.com/free-icon/ram_543279

Image: https://en.wikipedia.org/wiki/File:Amiga_1000DP.jpg
Icons: Freepik at Flaticon, e.g., https://www.flaticon.com/free-icon/ram_543279

What are the constraints that guided the architecture, design, and implementation of Deluxe Paint to handle the computational and memory constraints of the Amiga 1000?

# Pre-print

Historic Software Engineering:
Insights from Deluxe Paint for the Amiga

Giuseppe Destefanis[a], Yann-Gaël Guéhéneuc[b], Fabio Calefato[c]

[a] *University College London, UK*
[b] *Concordia University, Canada*
[c] *University of Bari, Italy*

**Abstract**

This article studies and discusses *Deluxe Paint*, a significant graphics program released in 1985 for the Amiga platform. Developed at a time when hardware constraints were severe, *Deluxe Paint* (*DPaint*) was written in C and designed to run on machines with only 256 KB of total memory, divided into video and CPU RAM, and a Motorola 68000 CPU at 7.09 MHz (PAL) or 7.16 MHz (NTSC).

We analyse and study the 17,001 lines of source code of this historic program **to understand the constraints that guided its architecture, design, and implementation** by developers who worked without widespread access to formal programming patterns, reference books, or online resources.

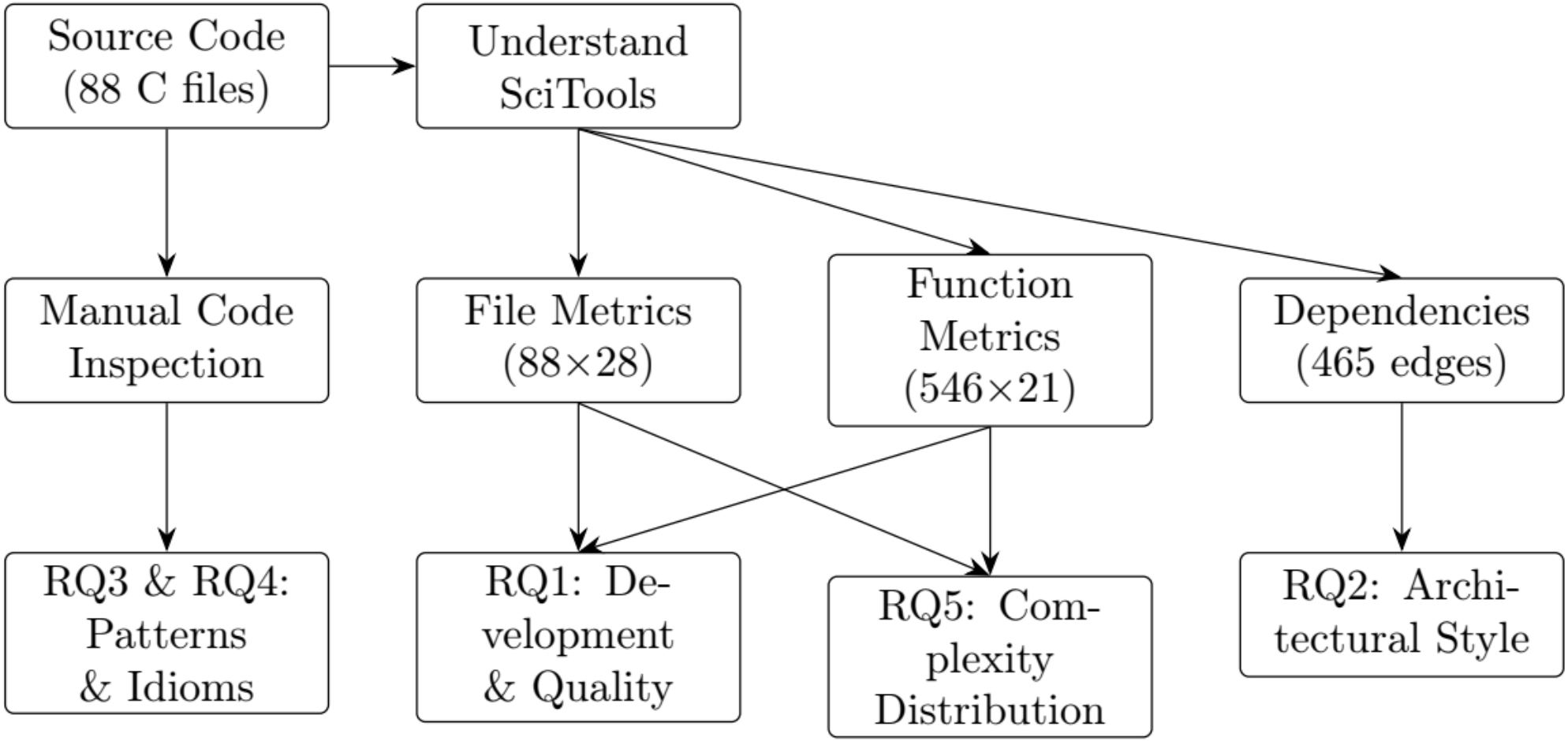
Our analysis covers the overall development, compilation, and quality of *DPaint*. We also discuss the architectural styles, design practices, and implementation idioms present in its source code, as well as its code complexity. We identify antecedents to 11 of the 23 Gang of Four design patterns, all implemented through C constructs, nine years before the GoF catalogue was published, plus seven Amiga platform-specific architectural idioms with no GoF counterpart.

We also show how *DPaint* exploited the compiler and the Amiga hardware for efficiency, including direct graphics manipulation and optimised memory access. These findings highlight the resourceful methods used by its developers to maximise performance on such constrained systems.

*Email addresses:* g.destefanis@ucl.ac.uk (Giuseppe Destefanis), yann-gael.gueheneuc@concordia.ca (Yann-Gaël Guéhéneuc), fabio.calefato@uniba.it (Fabio Calefato)

- RQ1: How and by whom was DPaint developed, and what was its overall quality?
- RQ2: What architectural styles can be observed in the source code of DPaint?
- RQ3: What design patterns are present in the source code of DPaint?
- RQ4: What coding idioms are present in the source code of DPaint?
- RQ5: How was complexity distributed across the different components of DPaint, and why?

# Method


Source Code (88 C files)
Understand SciTools
Manual Code Inspection
File Metrics (88×28)
Function Metrics (546×21)
Dependencies (465 edges)
RQ3 & RQ4: Patterns & Idioms
RQ1: Development & Quality
RQ5: Complexity Distribution
RQ2: Architectural Style

# Reproducibility

# Method in Details

- RQ1: How and by whom was DPaint developed, and what was its overall quality?
  - Qualitative analysis
    - Files and their naming conventions and contents
  - Descriptive statistics
  - Compilation and recompilation

# RQ1 Results

- ACTBMS.C
- AIRBRUSH.C
- BEND.C
- BITMAPS.C
- BLEND.C
- … (79 more files)
- SYMREQ.C
- SYSTEM.H
- TEXT.C
- TIMER.C
- UNPACKER.C

```
/*-----------------------------------------------------------------------*/
/*                                                                       */
/*                        prism.c -- Main program                        */
/*                                                                       */
/*-----------------------------------------------------------------------*/

#include <system.h>
#include <librarie\dosexten.h>
#include <prism.h>
#include <librarie\diskfont.h>
#include <dphook.h>

#define local static

extern void mainRefresh();
extern struct Menu *MainMenu;
extern BMOB curpenob,curbr;
extern void mainCproc(),mainPproc(),mainMproc(),DrawMode(),ShadMode();
extern Range cycles[];
extern void ZZZCursor(), UnZZZCursor(), Panic();

...

/*-----------------------------------------------------------------------*/
/* Yes, this is it, the MAIN PROGRAM:                                    */
/*-----------------------------------------------------------------------*/
main(argc,argv) int argc; char *argv[]; {
    DPInit(argc,argv);  /* initialization code */
    NewIMode(IM_draw);
    SetAirBRad(PMapX(24)); /* also brings in the drawing overlay*/
#ifdef DOWB
    if (loadAFile) LoadPic();
#endif
    CopyWNotice();
            loaded = YES;
            PListen();   /* this is the program */
    if (!haveWBench) BootIT();
    else CloseDisplay();
    }
```

# Original name

# RQT Results

- ACTBMS.C
- AIRBRUSH.C
- BEND.C
- BITMAPS.C
- BLEND.C
- … (79 more files)
- SYMREQ.C
- SYSTEM.H
- TEXT.C
- TIMER.C
- UNPACKER.C

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/*                        prism.c -- Main program                     */
/*                                                                    */
/*--------------------------------------------------------------------*/

#include <system.h>
#include <librarie\dosexten.h>
#include <prism.h>
#include <librarie\diskfont.h>
#include <dphook.h>

#define local static

extern void mainRefresh();
extern struct Menu *MainMenu;
extern BMOB curpenob,curbr;
extern void mainCproc(),mainPproc(),mainMproc(),DrawMode(),ShadMode();
extern Range cycles[];
extern void ZZZCursor(), UnZZZCursor(), Panic();

...

/*--------------------------------------------------------------------*/
/* Yes, this is it, the MAIN PROGRAM:                                 */
/*--------------------------------------------------------------------*/
main(argc,argv) int argc; char *argv[]; {
    DPInit(argc,argv);  /* initialization code */
    NewIMode(IM_draw);
    SetAirBRad(PMapX(24)); /* also brings in the drawing overlay*/
#ifdef DOWB
    if (loadAFile) LoadPic();
#endif
    CopyWNotice();
            loaded = YES;
            PListen();   /* this is the program */
    if (!haveWBench) BootIT();
    else CloseDisplay();
    }
```

# Original name
# Busy pointer

- ACTBMS.C
- AIRBRUSH.C
- BEND.C
- BITMAPS.C
- BLEND.C
- … (79 more files)
- SYMREQ.C
- SYSTEM.H
- TEXT.C
- TIMER.C
- UNPACKER.C

```
/*-----------------------------------------------------------------------*/
/*                                                                       */
/*                        prism.c -- Main program                        */
/*                                                                       */
/*-----------------------------------------------------------------------*/

#include <system.h>
#include <librarie\dosexten.h>
#include <prism.h>
#include <librarie\diskfont.h>
#include <dphook.h>

#define local static

extern void mainRefresh();
extern struct Menu *MainMenu;
extern BMOB curpenob,curbr;
extern void mainCproc(),mainPproc(),mainMproc(),DrawMode(),ShadMode();
extern Range cycles[];
extern void ZZZCursor(), UnZZZCursor(), Panic();

...

/*-----------------------------------------------------------------------*/
/* Yes, this is it, the MAIN PROGRAM:                                    */
/*-----------------------------------------------------------------------*/
main(argc,argv) int argc; char *argv[]; {
    DPInit(argc,argv);  /* initialization code */
    NewIMode(IM_draw);
    SetAirBRad(PMapX(24)); /* also brings in the drawing overlay*/
#ifdef DOWB
    if (loadAFile) LoadPic();
#endif
    CopyWNotice();
            loaded = YES;
            PListen();   /* this is the program */
    if (!haveWBench) BootIT();
    else CloseDisplay();
    }
```

# Original name
# Busy pointer
# Infinite loop

- AIRBRUSH.C
- BEND.C
- BITMAPS.C
- BLEND.C
- … (79 more files)
- SYMREQ.C
- SYSTEM.H
- TEXT.C
- TIMER.C
- UNPACKER.C

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/*                        prism.c -- Main program                     */
/*                                                                    */
/*--------------------------------------------------------------------*/

#include <system.h>
#include <librarie\dosexten.h>
#include <prism.h>
#include <librarie\diskfont.h>
#include <dphook.h>

#define local static

extern void mainRefresh();
extern struct Menu *MainMenu;
extern BMOB curpenob,curbr;
extern void mainCproc(),mainPproc(),mainMproc(),DrawMode(),ShadMode();
extern Range cycles[];
extern void ZZZCursor(), UnZZZCursor(), Panic();

...

/*--------------------------------------------------------------------*/
/* Yes, this is it, the MAIN PROGRAM:                                 */
/*--------------------------------------------------------------------*/
main(argc,argv) int argc; char *argv[]; {
    DPInit(argc,argv);  /* initialization code */
    NewIMode(IM_draw);
    SetAirBRad(PMapX(24)); /* also brings in the drawing overlay*/
#ifdef DOWB
    if (loadAFile) LoadPic();
#endif
    CopyWNotice();
            loaded = YES;
            PListen();   /* this is the program */
    if (!haveWBench) BootIT();
    else CloseDisplay();
    }
```

# Original name

# Busy pointer

# Infinite loop

- AIRBRUSH.C
- BEND.C
- BITMAPS.C
- BLEND.C
- … (79 more files)
- SYMREQ.C
- SYSTEM.H
- TEXT.C
- TIMER.C

# Can be closed

```
/*-----------------------------------------------------------------------*/
/*                                                                       */
/*                        prism.c -- Main program                        */
/*                                                                       */
/*-----------------------------------------------------------------------*/

#include <system.h>
#include <librarie\dosexten.h>
#include <prism.h>
#include <librarie\diskfont.h>
#include <dphook.h>

#define local static

extern void mainRefresh();
extern struct Menu *MainMenu;
extern BMOB curpenob,curbr;
extern void mainCproc(),mainPproc(),mainMproc(),DrawMode(),ShadMode();
extern Range cycles[];
extern void ZZZCursor(), UnZZZCursor(), Panic();

...

/*-----------------------------------------------------------------------*/
/* Yes, this is it, the MAIN PROGRAM:                                    */
/*-----------------------------------------------------------------------*/
main(argc,argv) int argc; char *argv[]; {
    DPInit(argc,argv);  /* initialization code */
    NewIMode(IM_draw);
    SetAirBRad(PMapX(24)); /* also brings in the drawing overlay*/
#ifdef DOWB
    if (loadAFile) LoadPic();
#endif
    CopyWNotice();
            loaded = YES;
            PListen();  /* this is the program */
    if (!haveWBench) BootIT();
    else CloseDisplay();
    }
```

# RQ1 Results

| Metric | Min. | Avg. | Max. | Sum |
|---|---|---|---|---|
| Number of Files | – | – | – | 89 |
| Lines of code | 10 | 193 | 1,337 | 17,001 |
| Cyclomatic complexity | 0 | 14 | 117 | 1,223 |
| Cyclomatic complexity | 0 | 8.6 | 74 | – |
| Nesting | 0 | 2 | 4 | – |

- 77 C source files
  - + 11 C headers
  - + 1 overlay description file
- Total size of 17,001 LOC
- File `PALETTE.C` has 1,337 LOC
- File `PALETTE.C` has 117 CC
- File `CHPROC.C` has the max avg. CC of 8.6

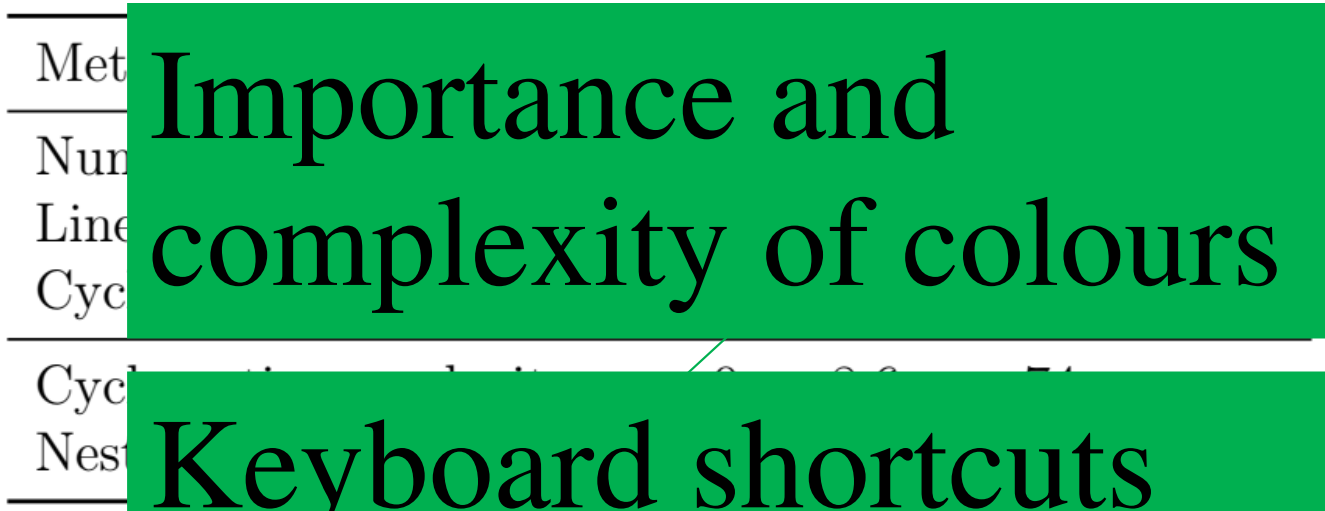
# RQ1 Results

| Met | | | | |
|---|---|---|---|---|
| Num | | | | |
| Line | | | | |
| Cyc | | | | |
| Cyclomatic complexity | 0 | 8.6 | 74 | – |
| Nesting | 0 | 2 | 4 | – |

**Importance and complexity of colours**

- 77 C source files
  - + 11 C headers
  - + 1 overlay description file
- Total size of 17,001 LOC
- File `PALETTE.C` has 1,337 LOC
- File `PALETTE.C` has 117 CC
- File `CHPROC.C` has the max avg. CC of 8.6

# RQ1 Results




- 77 C source files
  - + 11 C headers
  - + 1 overlay description file
- Total size of 17,001 LOC
- File `PALETTE.C` has 1,337 LOC
- File `PALETTE.C` has 117 CC
- File `CHPROC.C` has the max avg. CC of 8.6

# RQ1 Results

- Six contributors
  - Three clearly names authors
    - Dan Silva
    - Jerry Morrison
    - Steve Shaw
    - Gordon Knopes
  - Two “unknown” authors
    - `mrp` in `DPIFF.C`
    - `tomc` in `MAKEICON.C`
- Between 1985/06/22 and 1985/11/07
  - Less than 5 months!

# RQ1 Results

Small team and less rigorous attributions

- Six contributors
  - Three clearly names authors
    - Dan Silva
    - Jerry Morrison
    - Steve Shaw
    - Gordon Knopes
  - Two “unknown” authors
    - `mrp` in `DPIFF.C`
    - `tomc` in `MAKEICON.C`
- Between 1985/06/22 and 1985/11/07
  - Less than 5 months!

# RQ1 Results

Small team and less rigorous attributions

Little tech. doc.
No StackOverflow
Not even the Internet!

- Six contributors
  - Three clearly names a
    - Dan Silva
    - Jerry Morrison
    - Steve Shaw
    - Gordon Knopes
  - Two “unknown” authors
    - `mrp` in `DPIFF.C`
    - `tomc` in `MAKEICON.C`
- Between 1985/06/22 and 1985/11/07
  - Less than 5 months!

# RQ1 Results

- 8+3 naming convention
  - DRI CP/M and Microsoft DOS (MS-DOS)
- Compiled with Lattice C
  - Overlay system
- Recompiled with SAS/C
  - Only warnings in 11 categories

# RQ1 Results

Cross-development

- 8+3 naming convention
  - DRI CP/M and Microsoft DOS (MS-DOS)
- Compiled with Lattice C
  - Overlay system
- Recompiled with SAS/C
  - Only warnings in 11 categories

```
ROOT astartup.o, prism.o, menu.o, chproc.o, curbrush.o*
paintw.o, modes.o, geom.o, conic.o*
psym.o, lfmult.o, pane.o*
mainmag.o, magops.o, magwin.o, magbits.o*
pgraph.o, bitmaps.o, bmob.o*
blitops.o, maskblit.o, dalloc.o, box.o*
dispnum.o, keyjunk.o, distance.o, mousebut.o, ctrpan.o*
penmenu.o, actbms.o, cursor.o, cursbms.o, dotbms.o*
message.o, ccycle.o, reqcom.o, timer.o, brxform.o
OVERLAY
initovs.o, dpinit.o, dpinit2.o, fontinit.o, bootit.o, hook.o,
fill.o, airbrush.o, random.o, polyh.o, blend.o, shade.o, polyf.o,
    ↪ clip.o, text.o
rot90.o, stretch.o, rotate.o, shear.o, remap.o, bend.o, print.o,
    ↪ spare.o
dopalett.o, palette.o, palbms.o hsv.o
dosymreq.o,symreq.o
dpio.o, fnreq.o, fnbms.o, dpiff.o
*initread.o, ilbmr.o, unpacker.o,iffr.o
*initwrite.o, ilbmw.o, packerf.o, copymem.o, iffw.o, makeicon.o
#
LIBRARY amiga.lib, lc.lib, debug.lib
TO prism
```

```
ROOT astartup.o, prism.o, menu.o
paintw.o, modes.o, geom.o, conic
psym.o, lfmult.o, pane.o*
mainmag.o, magops.o, magwin.o, m
pgraph.o, bitmaps.o, bmob.o*
blitops.o, maskblit.o, dalloc.o,
dispnum.o, keyjunk.o, distance.o
penmenu.o, actbms.o, cursor.o, cursbms.o, dotbms.o*
message.o, ccycle.o, reqcom.o, timer.o, brxform.o
OVERLAY
initovs.o, dpinit.o, dpinit2.o, fontinit.o, bootit.o, hook.o,
fill.o, airbrush.o, random.o, polyh.o, blend.o, shade.o, polyf.o,
    ↪ clip.o, text.o
rot90.o, stretch.o, rotate.o, shear.o, remap.o, bend.o, print.o,
    ↪ spare.o
dopalett.o, palette.o, palbms.o hsv.o
dosymreq.o,symreq.o
dpio.o, fnreq.o, fnbms.o, dpiff.o
*initread.o, ilbmr.o, unpacker.o,iffr.o
*initwrite.o, ilbmw.o, packerf.o, copymem.o, iffw.o, makeicon.o
#
LIBRARY amiga.lib, lc.lib, debug.lib
TO prism
```

# Linker directives for dynamic (un)loading (cf. Smalltalk, Java)

```
ROOT astartup.o, prism.o, menu.o
paintw.o, modes.o, geom.o, conic
psym.o, lfmult.o, pane.o*
mainmag.o, magops.o, magwin.o, m
pgraph.o, bitmaps.o, bmob.o*
blitops.o, maskblit.o, dalloc.o,
dispnum.o, keyjunk.o, distance.o
penmenu.o, actbms.o, cursor.o, cursbms.o, dotbms.o*
message.o, ccycle.o, reqcom.o, timer.o, brxform.o
OVERLAY
initovs.o, dpinit.o, dpinit2.o,
fill.o, airbrush.o, random.o, po
    ↪ clip.o, text.o
rot90.o, stretch.o, rotate.o, shear.o, remap.o, bend.o, print.o,
    ↪ spare.o
dopalett.o, palette.o, palbms.o hsv.o
dosymreq.o,symreq.o
dpio.o, fnreq.o, fnbms.o, dpiff.o
*initread.o, ilbmr.o, unpacker.o,iffr.o
*initwrite.o, ilbmw.o, packerf.o, copymem.o, iffw.o, makeicon.o
#
LIBRARY amiga.lib, lc.lib, debug.lib
TO prism
```

Linker directives for dynamic (un)loading (cf. Smalltalk, Java)

Original name

# RQ1 Results

| Number | Definition |
|---|---|
| 62 | constant number out of range for type "*type*"<br>Valid range is *low* to *high* |
| 85 | return value mismatch for function "*name*"<br>Expecting "*type1*", found "*type2*" |
| 100 | no prototype declared for function "*name*" |
| 132 | extra tokens after valid preprocessor directive |
| 154 | no prototype declared for function pointer |
| 155 | no statement after label |
| 161 | no prototype declared at definition for function "*name*" |
| 169 | incompatible operands of conditional operator (?:)<br>"*type1*" conflicts with "*type2*" |
| 181 | "*name*" was declared both static and external<br>See line *number* file "*filename*" |
| 217 | macro invocation may call function multiple times |
| 224 | item "*name*" already defined<br>See line *number* file "*filename*" |

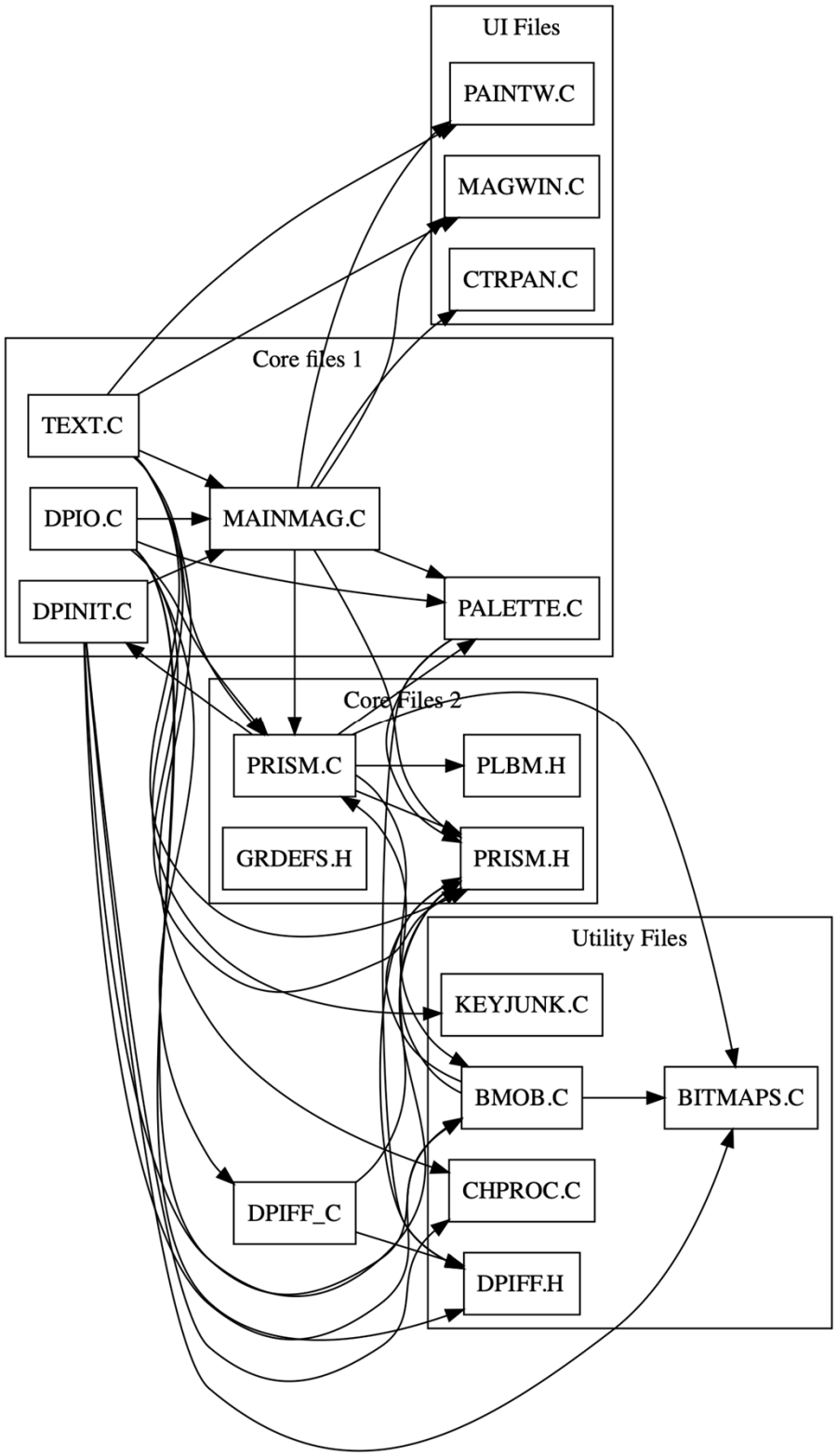
- Pre-ANSI C
  - Ill-defined primitive types, like `short`
  - Struct declaration collisions, like `Point`
  - Lack of function (pointer) declarations

# Method in Details

- RQ2: What architectural styles can be observed in the source code of DPaint?
  - Network analysis
    - File-to-file dependencies
  - Graph-related measures

# RQ2 Results

- Centralised coordination
  - `PRISM.C` and `PRISM.H`
- Layered abstraction
  - UI vs. Core files vs. Utils
- Modular design
  - High cohesion



# RQ2 Results

| File | Betw. | File | In-deg. |
|---|---|---|---|
| PRISM.C | 0.137 | SYSTEM.H | 58 |
| PAINTW.C | 0.096 | PRISM.H | 48 |
| MENU.C | 0.074 | PRISM.C | 27 |
| MODES.C | 0.067 | PAINTW.C | 19 |
| MAINMAG.C | 0.043 | CURSOR.C | 14 |
| DPINIT.C | 0.034 | CCYCLE.C | 14 |
| DPIO.C | 0.030 | MODES.C | 14 |
| PGRAPH.C | 0.023 | MAINMAG.C | 14 |
| BMOB.C | 0.020 | BMOB.C | 13 |
| CURBRUSH.C | 0.017 | PGRAPH.C | 13 |

- Centrality analysis
  - Betweeness
    - Shortest paths going “through” a node
  - In-degree
    - Number of edges pointing on a node

# RQ2 Results

- Community detection
  - Greedy modularity optimisation algorithm
    - Clauset et al., 2004
  - Five communities
    - System core (17 files)
    - Drawing utilities (16 files)
    - Interaction and rendering (15 files)
    - Bitmap operations (13 files)
    - File I/O and colour (9 files)

# RQ2 Results

- Robustness analysis
  - Resilience to loss of bridge files
  - Sensitive to loss of foundational files
    - `PRISM.H` and `SYSTEM.H`
  - Practices
    - Hardware awareness, e.g., Blitter
    - Memory optimisation, e.g., bit fields
    - Performance focus, e.g., macros
    - Maintainability, e.g., high comment ratios

# Method in Details

- RQ3: What design patterns are present in the source code of DPaint?
  - Manual inspection
    - Function signatures
    - Struct definitions
    - Macro definitions
    - Dispatch tables
    - Function pointers

# RQ3 Results

| Family | Pattern | Primary Evidence |
|---|---|---|
| Behavioural | Command (8-slot procs) | `PAINTW.C`: 28–30 |
| | Command (Undo tags) | `MAINMAG.C`: 71, `PRISM.H`: 396 |
| | Command (Menu dispatch) | `MENU.C`: 159, 391 |
| | Observer (Pane callbacks) | `PANE.C`, `CCYCLE.C` |
| | State (IMode FSM) | `PAINTW.C`: 237–271 |
| | Strategy (Pixel writers) | `PGRAPH.C`: 100–102 |
| | Strategy (Geo. primitives) | `GEOM.C`, `CONIC.C` |
| | Template Method | `PAINTW.C`: 166–209, `MODES.C` |
| Creational | Factory (Bitmap/Memory) | `BITMAPS.C`, `DALLOC.C` |
| | Factory (Pen dispatch) | `CURBRUSH.C`: 141 |
| | Prototype (Bitmap clone) | `BITMAPS.C`: 127–138 |
| | Prototype (Undo swap) | `MAINMAG.C`: 102–110 |
| | Singleton/Monostate | `PRISM.C` |
| Structural | Adapter (Coordinates) | `PRISM.H`: 110–115 |
| | Adapter (IFF Format) | `DPIFF.C`, `ILBMR/W.C` |
| | Decorator (BMOB layers) | `PRISM.H`: 123–157 |
| | Decorator (IMode flags) | `PRISM.H`: 306–333 |
| | Façade (Graphics) | `PGRAPH.C` |
| | Façade (Blitter) | `BLITOPS.C`, `MASKBLIT.C` |
| | Façade (Bitmap Mem.) | `BITMAPS.C` |
| | Façade (Windowing) | `PANE.C` |

# Method in Details

- RQ4: What coding idioms are present in the source code of DPaint?
  - Manual inspection
    - Low-level programming practices
  - Code density and documentation
  - Coupling and stability measures

# RQ4 Results

- Code Density and Documentation
  - 88 files, mean of 0.24
- Function Signatures
  - 21% of the functions take 0 parameter
  - 39% 1-2, 29% 3-5, and 11% 6+
- Preprocessor Usage
  - 88 files, mean of 0.09
- Coupling and Instability
  - Layered architecture

# RQ4 Results

- Architectural idioms

| | | |
|---|---|---|
| Architectural | Code Overlay System | `PRISM.txt`, `PRISM.H`: 409 |
| | Cooperative Resource Sharing | `PRISM.C`: 343, `DPIO.C`: 42 |
| | Graphics Context Stack | `PGRAPH.C`: 38–52 |
| | Hardware Register Abstraction | `PRISM.H`: 382–390 |
| | Inter-process Service Locator | `HOOK.C`, `DPHOOK.H` |
| | Per-object Save-under | `BMOB.C`: 311–403 |
| | Two-buffer Undo | `MAINMAG.C`: 58–110 |

# Method in Details

- RQ5: How was complexity distributed across the different components, and why?
  - Complexity and categorisation
  - Correlation and risk analysis

# RQ5 Results

- Understand SciTools

| File | Sum CC | LOC | Max CC | Avg CC | Max Nest |
|---|---|---|---|---|---|
| PALETTE.C | 117 | 1,337 | 23 | 3.8 | 3 |
| CHPROC.C | 95 | 253 | 74 | 8.6 | 3 |
| MODES.C | 93 | 457 | 6 | 1.3 | 1 |
| PANE.C | 62 | 417 | 13 | 2.7 | 4 |
| MAINMAG.C | 58 | 451 | 6 | 1.8 | 2 |
| MENU.C | 53 | 454 | 22 | 3.3 | 2 |
| PAINTW.C | 47 | 429 | 5 | 2.0 | 2 |
| DPIO.C | 41 | 352 | 6 | 2.7 | 4 |
| BMOB.C | 37 | 417 | 11 | 2.5 | 2 |
| PGRAPH.C | 37 | 313 | 6 | 1.6 | 3 |

| Category | Sum CC | LOC | Files | Avg CC/File |
|---|---|---|---|---|
| UI / Interaction | 472 | 4,389 | 16 | 29.5 |
| Palette / Colour | 208 | 2,299 | 8 | 26.0 |
| Bitmap / Blitter | 161 | 3,089 | 13 | 12.4 |
| Transforms | 131 | 1,206 | 7 | 18.7 |
| System / Init | 110 | 1,557 | 15 | 7.3 |
| Drawing / Graphics | 96 | 1,284 | 8 | 12.0 |
| File I/O | 45 | 2,058 | 10 | 4.5 |

- Files
  - [1, 21.8, 117]
- Categories
  - [45, 174.7, 472]

# RQ5 Results

- Understand SciTools

| File | Sum CC | LOC | Max CC | Avg CC | Max Nest |
|---|---|---|---|---|---|
| PALETTE.C | 117 | 1,337 | 23 | 3.8 | 3 |
| CHPROC.C | 95 | 253 | 74 | 8.6 | 3 |
| MODES.C | 93 | 457 | 6 | 1.3 | 1 |
| PANE.C | 62 | 417 | 13 | 2.7 | 4 |
| MAINMAG.C | 58 | 451 | 6 | 1.8 | 2 |
| MENU.C | 53 | 454 | 22 | 3.3 | 2 |
| PAINTW.C | 47 | 429 | 5 | 2.0 | 2 |
| DPIO.C | 41 | 352 | 6 | 2.7 | 4 |
| BMOB.C | 37 | 417 | 11 | 2.5 | 2 |
| PGRAPH.C | 37 | 313 | 6 | 1.6 | 3 |

| Category | Sum CC | LOC | Files | Avg CC/File |
|---|---|---|---|---|
| UI / Interaction | 472 | 4,389 | 16 | 29.5 |
| Palette / Colour | 208 | 2,299 | 8 | 26.0 |
| Bitmap / Blitter | 161 | 3,089 | 13 | 12.4 |
| Transforms | 131 | 1,206 | 7 | 18.7 |
| System / Init | 110 | 1,557 | 15 | 7.3 |
| Drawing / Graphics | 96 | 1,284 | 8 | 12.0 |
| File I/O | 45 | 2,058 | 10 | 4.5 |

- Files
  - [1, 21.8, 117]
- Categories
  - [45, 174.7, 472]

# RQ5 Results

- Understand SciTools

| File | Sum CC | LOC | Max CC | Avg CC | Max Nest |
|---|---|---|---|---|---|
| PALETTE.C | 117 | 1,337 | 23 | 3.8 | 3 |
| CHPROC.C | 95 | 253 | 74 | 8.6 | 3 |
| MODES.C | 93 | 457 | 6 | 1.3 | 1 |
| PANE.C | 62 | 417 | 13 | 2.7 | 4 |
| MAINMAG.C | 58 | 451 | 6 | 1.8 | 2 |
| MENU.C | 53 | 454 | 22 | 3.3 | 2 |
| PAINTW.C | 47 | 429 | 5 | 2.0 | 2 |
| DPIO.C | 41 | 352 | 6 | 2.7 | 4 |
| BMOB.C | 37 | 417 | 11 | 2.5 | 2 |
| PGRAPH.C | 37 | 313 | 6 | 1.6 | 3 |

| Category | Sum CC | LOC | Files | Avg CC/File |
|---|---|---|---|---|
| UI / Interaction | 472 | 4,389 | 16 | 29.5 |
| Palette / Colour | 208 | 2,299 | 8 | 26.0 |
| Bitmap / Blitter | 161 | 3,089 | 13 | 12.4 |
| Transforms | 131 | 1,206 | 7 | 18.7 |
| System / Init | 110 | 1,557 | 15 | 7.3 |
| Drawing / Graphics | 96 | 1,284 | 8 | 12.0 |
| File I/O | 45 | 2,058 | 10 | 4.5 |

- Files
  - [1, 21.8, 117]
- Categories
  - [45, 174.7, 472]

# RQ5 Results

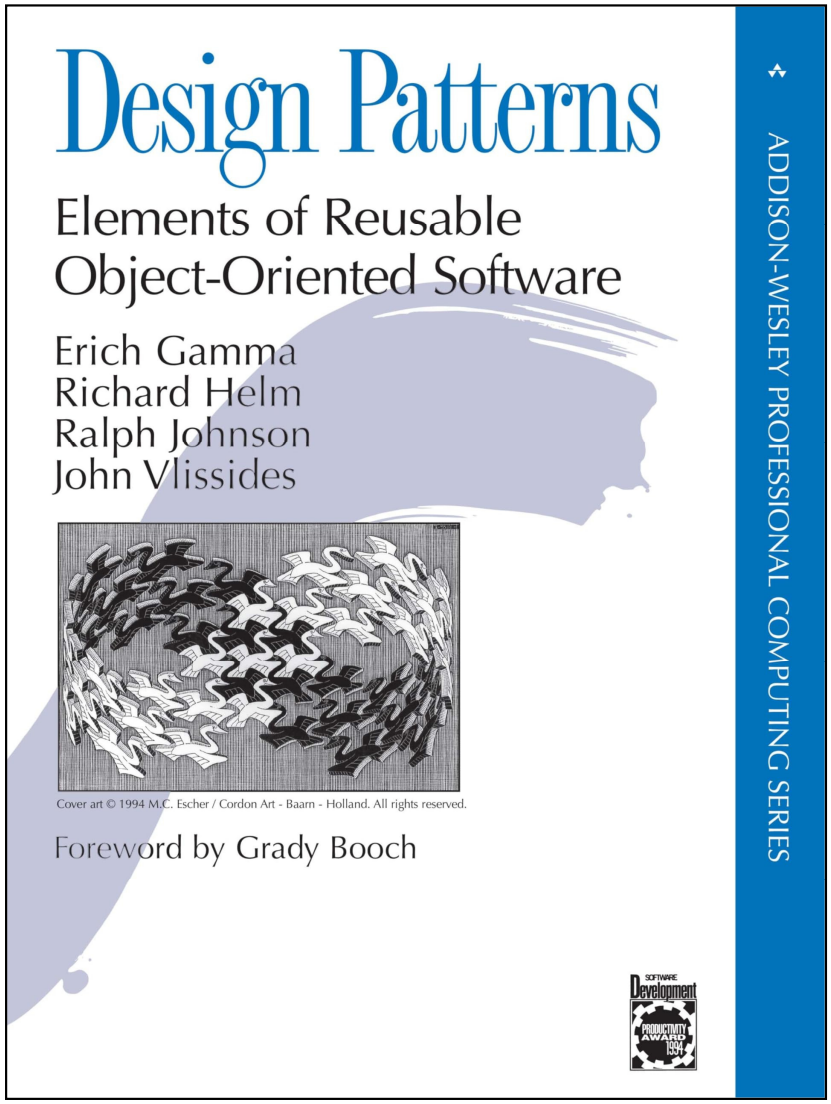
- Understand SciTools

| File | Sum CC | LOC | Max CC | Avg CC | Max Nest |
|---|---|---|---|---|---|
| PALETTE.C | 117 | 1,337 | 23 | 3.8 | 3 |
| CHPROC.C | 95 | 253 | 74 | 8.6 | 3 |
| MODES.C | 93 | 457 | 6 | 1.3 | 1 |
| PANE.C | 62 | 417 | 13 | 2.7 | 4 |
| MAINMAG.C | 58 | 451 | 6 | 1.8 | 2 |
| MENU.C | 53 | 454 | 22 | 3.3 | 2 |
| PAINTW.C | 47 | 429 | 5 | 2.0 | 2 |
| DPIO.C | 41 | 352 | 6 | 2.7 | 4 |
| BMOB.C | 37 | 417 | 11 | 2.5 | 2 |
| PGRAPH.C | 37 | 313 | 6 | 1.6 | 3 |

| Category | Sum CC | LOC | Files | Avg CC/File |
|---|---|---|---|---|
| UI / Interaction | 472 | 4,389 | 16 | 29.5 |
| Palette / Colour | 208 | 2,299 | 8 | 26.0 |
| Bitmap / Blitter | 161 | 3,089 | 13 | 12.4 |
| Transforms | 131 | 1,206 | 7 | 18.7 |
| System / Init | 110 | 1,557 | 15 | 7.3 |
| Drawing / Graphics | 96 | 1,284 | 8 | 12.0 |
| File I/O | 45 | 2,058 | 10 | 4.5 |

- Files
  - [1, 21.8, 117]
- Categories
  - [45, 174.7, 472]

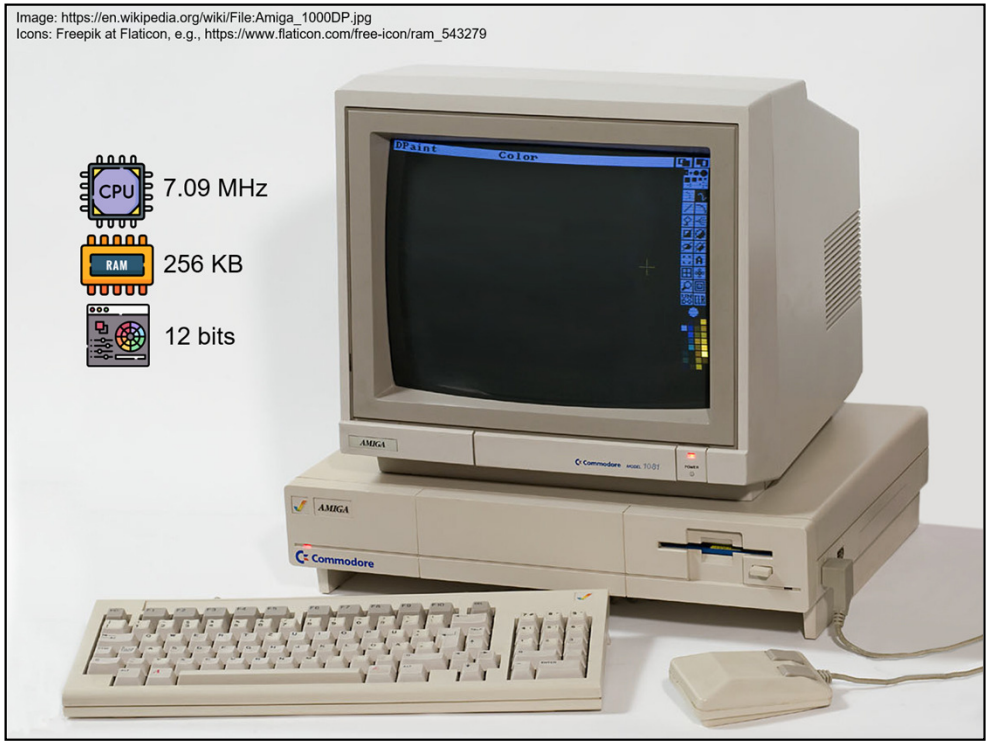
# Discussions




"DPaint's developers independently arrived at structural solutions that the GoF would formalise nine years later, using function pointers, struct embedding, and dispatch tables in place of classes and virtual methods."

# Discussions

“DPaint’s developers [created] modular, low-coupling designs where each file has a clear responsibility. The hub-and-spoke architecture centred on `PRISM.C/H` emerged [...] from the practical need to share scarce resources [...] without duplicating state.”



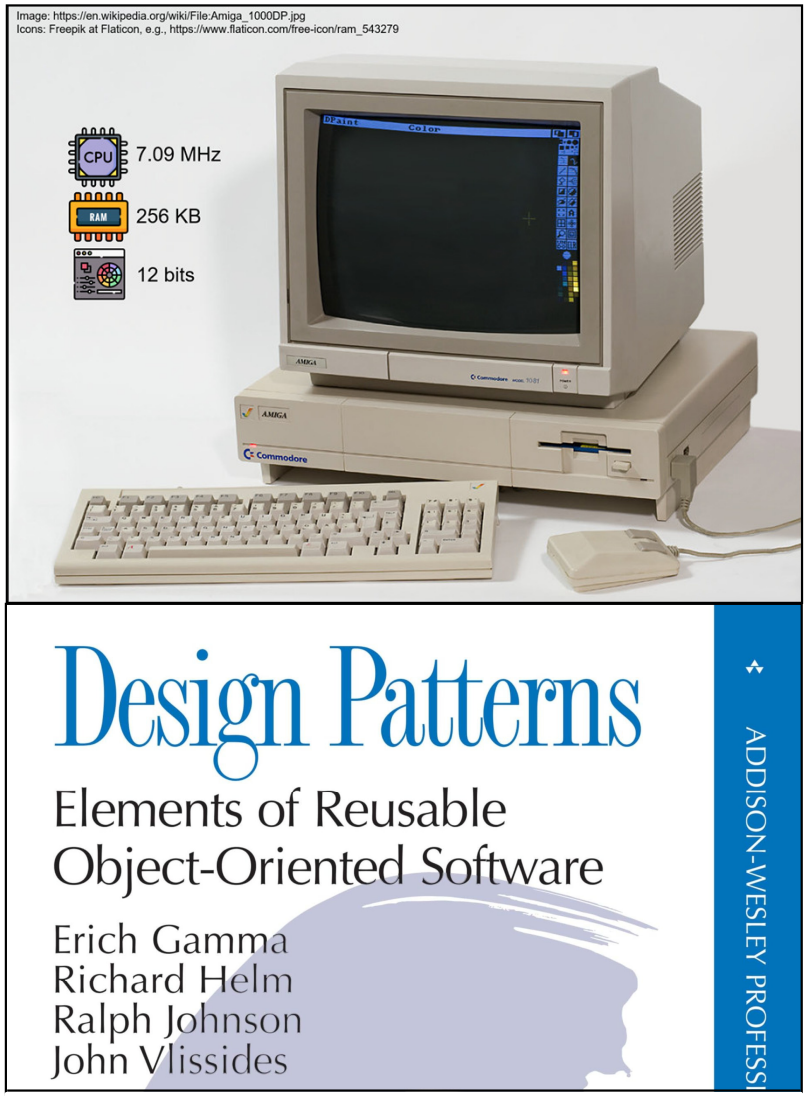

https://www.flickr.com/photos/blakespot/4364963257

# Discussions

"DPaint's developers [created] modular, low-coupling designs where each file has a clear responsibility. The hub-and-spoke architecture centred on `PRISM.C/H` emerged [...] from the practical need to share scarce resources [...] without duplicating state."


https://www.flickr.com/photos/blakespot/4364963257

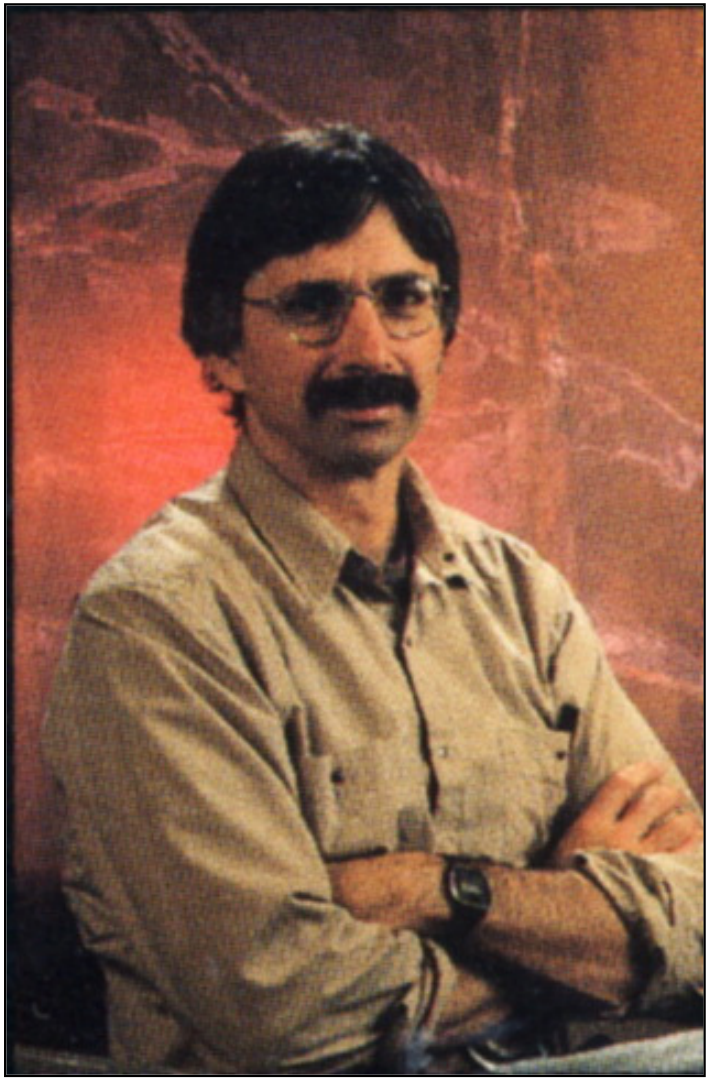
# Discussions




"[T]he absence of […] Composite, Iterator, and Flyweight, is not accidental: tree structures consume pointer overhead, iterator abstractions add indirection costs, and shared intrinsic-state pools require management overhead."

# Discussions

"[A] single-developer project can achieve a high degree of consistency."



+ EA's 1983 ad series "Can a computer make you cry?"
+ EA's attempt to promote developers as rock starts

https://computerhistory.org/blog/electronic-arts-deluxepaint-early-source-code/

# Discussions

“Without version control, IDEs, linters, or automated testing, DPaint source code compiles with only warnings on a modern compiler and runs correctly on original hardware. The code’s quality is evidenced by its commercial success and […] compilability 40 years later.”

https://www.classic-computers.org.nz/collection/ibm-at.htm

# Lessons Learned

- Code overlay system
- Cooperative resource sharing
- C struct field names onto memory-mapped I/O registers
- Virtual machine-based class (un)loading
  - OSGi
- Resource pooling in IoT devices
- Hardware Abstract Layers in Linux and embedded systems

# Related Work

# Related Work

2023 IEEE Integrated STEM Education Conference (ISEC)

## Retrocomputing in Contemporary Integrative STEM Education

Zhemin Zhang
Rensselaer Polytechnic Institute, hisenzhang01@gmail.com

2023 IEEE Integrated STEM Education Conference (ISEC) | 979-8-3503-0001-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISEC57711.2023.10402127

***Abstract* – In a world of exponentially growing demand for computing power running into chip technology challenges, it takes both software and hardware backgrounds to design and constructs an optimal system. The existing computer system engineering curricula have drawbacks as they cannot reach the equilibrium between the simplicity of theory and complicated real-world practical systems. This article proposes an initiative to introduce retrocomputing to bridge the gap between hardware and software curricula. The core idea is to bring simple yet practical computer systems into the classroom, specifically consumer market microcomputers built upon microprocessor technology in the 1980s. The retrocomputing activities can be summarized into a three-stage collect-restore-build pathway with progressively demanding knowledge and problem-solving skill requirements for different levels, from middle school to college. In addition to hands-on experience, introducing retrocomputing as a hobby has several benefits, including developing self-motivated students, lifetime learning habits, and sustainability awareness. Challenges such as class organization and insufficient teaching resources exist, but there are workarounds.**

*Index Terms* - Electrical Engineering, Computer System Engineering, Integrative STEM, Retrocomputing

### I. Introduction

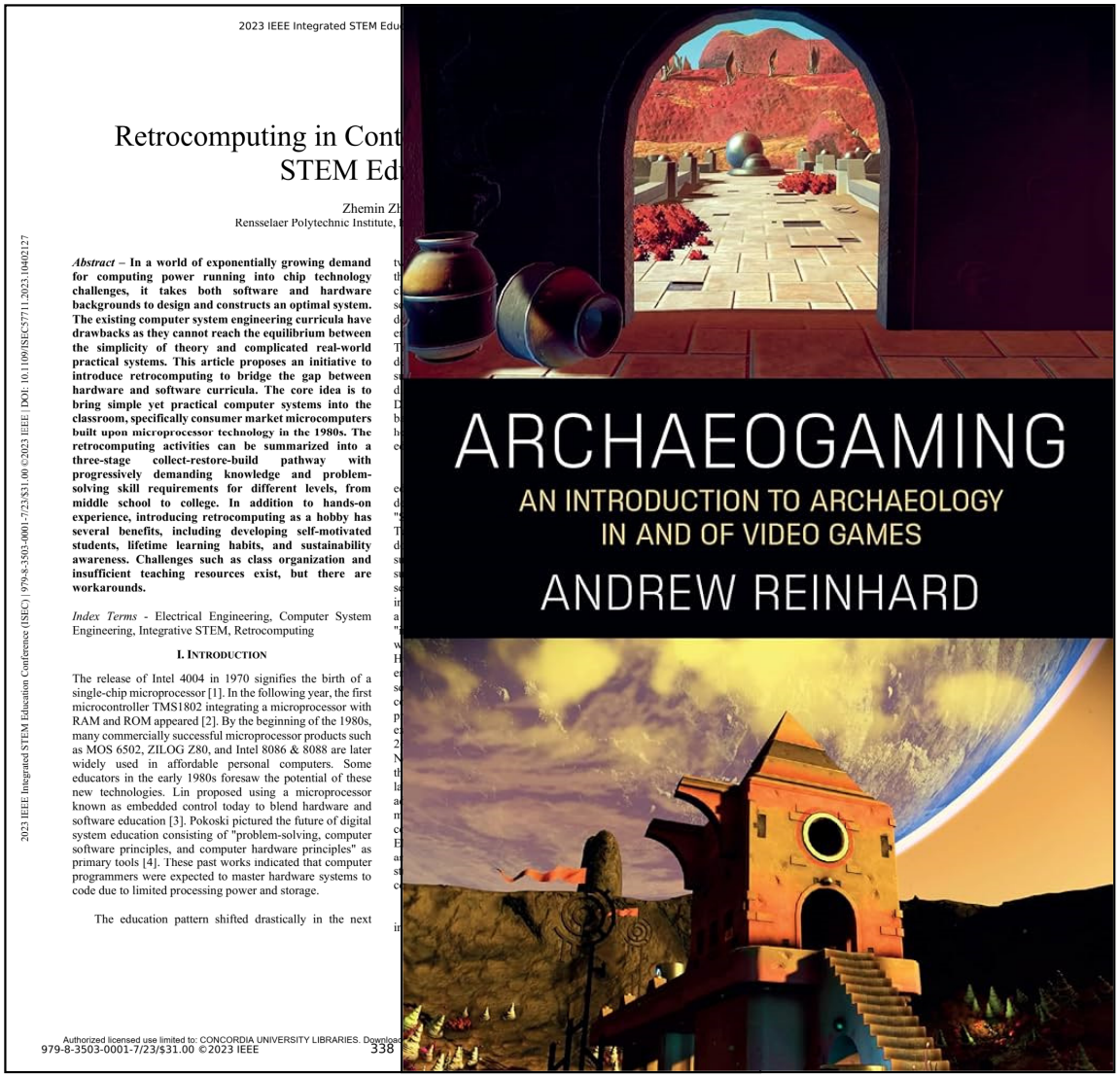
The release of Intel 4004 in 1970 signifies the birth of a single-chip microprocessor [1]. In the following year, the first microcontroller TMS1802 integrating a microprocessor with RAM and ROM appeared [2]. By the beginning of the 1980s, many commercially successful microprocessor products such as MOS 6502, ZILOG Z80, and Intel 8086 & 8088 are later widely used in affordable personal computers. Some educators in the early 1980s foresaw the potential of these new technologies. Lin proposed using a microprocessor known as embedded control today to blend hardware and software education [3]. Pokoski pictured the future of digital system education consisting of "problem-solving, computer software principles, and computer hardware principles" as primary tools [4]. These past works indicated that computer programmers were expected to master hardware systems to code due to limited processing power and storage.

The education pattern shifted drastically in the next twenty years: computer science (CS) students only learned the software aspect. They rarely touched circuit boards in class because the hardware performance was surplus to software requirements, thus no need to dive into low-level details. Furthermore, it is common to find electrical engineering (EE) students fearful of computer programming. Today, artificial intelligence's exponentially growing demand for computing power runs into chip technology and supply challenges, calling for rejoining of once-divided disciplines: we see examples of Google's TPU [5], Nvidia's DLSS [6], and Apple's M1 [7]. It takes software and hardware backgrounds to design and constructs an optimal system; hence, the industry embraces interdisciplinary expertise and educators again.

In response to such a trend, the concept of STEM education became widely accepted and practiced, despite its definition never reaching an agreement [8]. The word "STEM" may refer to a set of isolated subjects of Science, Technology, Engineering, and Mathematics or a unified domain that combines these subjects. The isolation of STEM subjects is common in high school curricula as different subjects are designed, developed, taught, and evaluated separately; in comparison, most college courses are interdisciplinary because they are closer to the real world. As a result, "Integrative STEM" was proposed, aiming to "intentionally" integrate the concepts from math and science with technology and engineering [8]. In the meantime, Higher education has started to offer computer system engineering programs that combine computer hardware and software topics to deliver a panoramic view for students. The core requirements for the computer system engineering program at Rensselaer Polytechnic Institute serve as an example [9]: Computer Components and Operations (ENGR-2610), in conjunction with Computer Architecture, Networks, and Operating Systems (ENGR-2660), emphasize the theory, while Embedded System (ENGR-2350) offers laboratory experience, both serving as prerequisites for advanced topics. To be specific, ENGR-2610 introduces the most fundamental concepts in computers such as combinational logic, latches, and finite state machines. ENGR-2660 covers assembly language, CPU architecture, and crude functions of operating systems. In ENGR 2350, students learn to program the MCS-51 microcontroller, commonly known as 8051.

Recent integrative STEM programs incorporate more intriguing topics for students than traditional ones. Some

# Related Work



2023 IEEE Integrated STEM Edu

Retrocomputing in Cont
STEM Ed

Zhemin Zh
Rensselaer Polytechnic Institute,

2023 IEEE Integrated STEM Education Conference (ISEC) | 979-8-3503-0001-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISEC57711.2023.10402127

*Abstract* – **In a world of exponentially growing demand for computing power running into chip technology challenges, it takes both software and hardware backgrounds to design and constructs an optimal system. The existing computer system engineering curricula have drawbacks as they cannot reach the equilibrium between the simplicity of theory and complicated real-world practical systems. This article proposes an initiative to introduce retrocomputing to bridge the gap between hardware and software curricula. The core idea is to bring simple yet practical computer systems into the classroom, specifically consumer market microcomputers built upon microprocessor technology in the 1980s. The retrocomputing activities can be summarized into a three-stage collect-restore-build pathway with progressively demanding knowledge and problem-solving skill requirements for different levels, from middle school to college. In addition to hands-on experience, introducing retrocomputing as a hobby has several benefits, including developing self-motivated students, lifetime learning habits, and sustainability awareness. Challenges such as class organization and insufficient teaching resources exist, but there are workarounds.**

*Index Terms* - Electrical Engineering, Computer System Engineering, Integrative STEM, Retrocomputing

## I. Introduction

The release of Intel 4004 in 1970 signifies the birth of a single-chip microprocessor [1]. In the following year, the first microcontroller TMS1802 integrating a microprocessor with RAM and ROM appeared [2]. By the beginning of the 1980s, many commercially successful microprocessor products such as MOS 6502, ZILOG Z80, and Intel 8086 & 8088 are later widely used in affordable personal computers. Some educators in the early 1980s foresaw the potential of these new technologies. Lin proposed using a microprocessor known as embedded control today to blend hardware and software education [3]. Pokoski pictured the future of digital system education consisting of "problem-solving, computer software principles, and computer hardware principles" as primary tools [4]. These past works indicated that computer programmers were expected to master hardware systems to code due to limited processing power and storage.

The education pattern shifted drastically in the next

# Related Work

2023 IEEE Integrated STEM Edu

## Retrocomputing in Cont
## STEM Ed

Zhemin Zh
Rensselaer Polytechnic Institute,

*Abstract* – **In a world of exponentially growing demand for computing power running into chip technology challenges, it takes both software and hardware backgrounds to design and constructs an optimal system. The existing computer system engineering curricula have drawbacks as they cannot reach the equilibrium between the simplicity of theory and complicated real-world practical systems. This article proposes an initiative to introduce retrocomputing to bridge the gap between hardware and software curricula. The core idea is to bring simple yet practical computer systems into the classroom, specifically consumer market microcomputers built upon microprocessor technology in the 1980s. The retrocomputing activities can be summarized into a three-stage collect-restore-build pathway with progressively demanding knowledge and problem-solving skill requirements for different levels, from middle school to college. In addition to hands-on experience, introducing retrocomputing as a hobby has several benefits, including developing self-motivated students, lifetime learning habits, and sustainability awareness. Challenges such as class organization and insufficient teaching resources exist, but there are workarounds.**

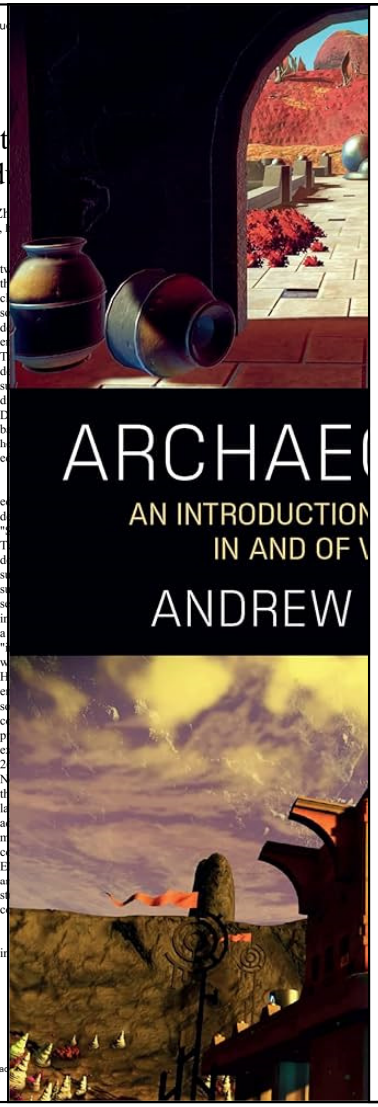
*Index Terms* - Electrical Engineering, Computer System Engineering, Integrative STEM, Retrocomputing

### I. Introduction

The release of Intel 4004 in 1970 signifies the birth of a single-chip microprocessor [1]. In the following year, the first microcontroller TMS1802 integrating a microprocessor with RAM and ROM appeared [2]. By the beginning of the 1980s, many commercially successful microprocessor products such as MOS 6502, ZILOG Z80, and Intel 8086 & 8088 are later widely used in affordable personal computers. Some educators in the early 1980s foresaw the potential of these new technologies. Lin proposed using a microprocessor known as embedded control today to blend hardware and software education [3]. Pokoski pictured the future of digital system education consisting of "problem-solving, computer software principles, and computer hardware principles" as primary tools [4]. These past works indicated that computer programmers were expected to master hardware systems to code due to limited processing power and storage.

The education pattern shifted drastically in the next

2023 IEEE Integrated STEM Education Conference (ISEC) | 979-8-3503-0001-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISEC57711.2023.10402127







*Journal of Electronic Gaming and Esports*, 2023, 1, 1-12
https://doi.org/10.1123/jege.2022-0041


Human Kinetics
ORIGINAL RESEARCH

## The Internet Is Not Forever: Challenges and Sustainability in Video Game Archiving and Preservation

**Brianna Dym,[1] Ellen Simpson,[2] Olivia Fong,[3] and Iibi striegl[2]**
[1]Roux Institute at Northeastern University, Portland, ME, USA; [2]University of Colorado Boulder, Boulder, CO, USA; [3]Independent Scholar, Albany, NY, USA

Video games are an increasingly significant cultural touchstone in people's everyday lives. However, preserving and archiving video games faces unique challenges, including intellectual property law, technology degradation, and the broader question of what it means to preserve a video game. In an exploratory study investigating sustainable game preservation practices, we spoke to 15 amateur game preservationists and hobbyists about their informal work with code, gaming consoles, and servers for online play. We found a lack of access to particular games during childhood or young adulthood led participants to seek out these games in other formats—such as emulated games they could play on other mediums (e.g., playing Nintendo games on your personal computer). Their nostalgia and the communities they found searching for these experiences inspired them to undertake archival work. Participants leveraged distributed knowledge across their communities to keep video games accessible for anyone interested in playing them. Considering these findings in the context of modern archival practices, we discuss what it means to archive a game, especially when that game is dependent on interactive, communal experiences, and what is potentially lost in current archival practices in contrast to informal, accidental archival work.

***Keywords:*** video games, digital archives, modding, romhacks

*I think with some games, Nintendo is very much like, oh, yeah, Mario World, we got ya. [ . . . ] But there's some games where it's either untranslated, like there's this game called Wagian Paradise. [ . . . ] And it's definitely not popular enough to warrant being put on Switch Online. Because like, who'd pay for that? But like, you know, I mean, I'd paid for that. —P15*

In 2023, Nintendo shuts down their eShop services for the 3DS and WiiU. This, along with Sony's announcement that they intended to shut down their online storefronts for the PlayStation 3 and PlayStation Vita, represents change for hobbyists who wish to play older games (Kuhnke, 2022). Although Sony eventually reversed their decision, the closure of digital marketplaces catering to older generations of consoles led to a loss of access for many gamers. In fact, as Oisin Kuhnke of *GameSpot* points out, with the closure of these virtual marketplaces, some games will be lost to players who wish to play the game legally and cannot afford to pay hundreds of dollars for a physical copy of the game cartridge (Kuhnke, 2022). Moreover, these physical cartridges will not last forever, and even the games preserved at official game archives, such as the Computer History Museum or the National Videogame Archive, run the risk of being lost due to the natural demagnetization of game storage on physical game cartridges and discs (Monnens, 2009). Archiving video games and consoles requires restoration work that is relatively new compared with preservation practices in mediums such as film or music (Kuhnke, 2022).

Video games exist in a complicated place in terms of archival work and cultural recognition—it is challenging to convince people who do not play games that they are an art form worth preserving (Kuhnke, 2022). Formal archival work is still determining what it can and cannot do in preserving video games as their code. The code itself is often proprietary and controlled by companies like Sony or Nintendo (Kraus & Donahue, 2012). Beyond formal archival work, many game hobbyists hack and modify game code (Newman, 2018) and pirate games (Newman, 2012) for personal collections, representing part of the effort toward culturally preserving games. Video game scholars have written extensively about the importance of hobbyist practices of modifying existing games—or "modding"—that copy and modify game code for hobbies like building fan games, making challenge runs, building new game content, and speed running (Consalvo, 2013; Murphy, 2013; Newman, 2012, 2018). Here, we note that video game companies are antagonistic toward game modding communities and modders themselves because of the *modding* of the game rather than any assumed commercialization of the modified game code. Modified games such as Pokemon Uranium (Frank, 2016b), Pokemon Prism (Machkovech, 2016), a fan remake of Metroid 2 (Frank, 2016a), and, more recently, a Grand Theft Auto mod all received Digital Millennium Copyright Act (DMCA) takedown notices. Nintendo has shut down people who share speedrunning videos on YouTube as well (Parlock, 2015).

As the field of game studies develops, we must investigate sustainable practices for preserving not just video games themselves but also the creative materials generated around and from video games as historical and cultural artifacts. In this article, we expand on prior work documenting the practices of hobbyists working with game code and digital emulators (Skold, 2018). Digital emulators are programs that operate on a computer that can play game code that an archivist, hobbyist, or gaming company has made available online. Due to the limitations of what archivists can and cannot legally do to preserve games and their derivative works, talking to game hobbyists and documenting their practices presents an opportunity to better understand best practices in preserving and sustaining digital games and their surrounding play.

In an exploratory study into sustainable game preservation practices, we present findings from 15 semistructured interviews

Dym (b.dym@northeastern.edu) is corresponding author.

# Related Work



## Retrocomputing in Cont
## STEM Ed

Zhemin Zh
Rensselaer Polytechnic Institute,

*Abstract* – **In a world of exponentially growing demand for computing power running into chip technology challenges, it takes both software and hardware backgrounds to design and constructs an optimal system. The existing computer system engineering curricula have drawbacks as they cannot reach the equilibrium between the simplicity of theory and complicated real-world practical systems. This article proposes an initiative to introduce retrocomputing to bridge the gap between hardware and software curricula. The core idea is to bring simple yet practical computer systems into the classroom, specifically consumer market microcomputers built upon microprocessor technology in the 1980s. The retrocomputing activities can be summarized into a three-stage collect-restore-build pathway with progressively demanding knowledge and problem-solving skill requirements for different levels, from middle school to college. In addition to hands-on experience, introducing retrocomputing as a hobby has several benefits, including developing self-motivated students, lifetime learning habits, and sustainability awareness. Challenges such as class organization and insufficient teaching resources exist, but there are workarounds.**

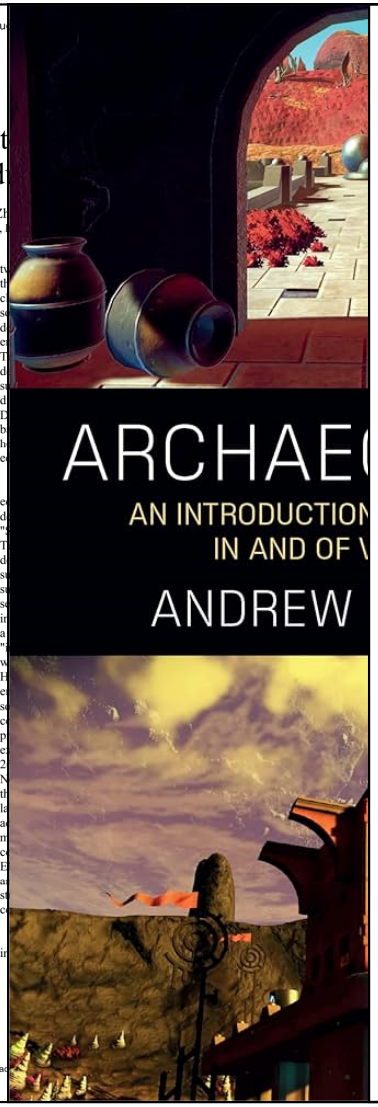
*Index Terms* - Electrical Engineering, Computer System Engineering, Integrative STEM, Retrocomputing

### I. Introduction

The release of Intel 4004 in 1970 signifies the birth of a single-chip microprocessor [1]. In the following year, the first microcontroller TMS1802 integrating a microprocessor with RAM and ROM appeared [2]. By the beginning of the 1980s, many commercially successful microprocessor products such as MOS 6502, ZILOG Z80, and Intel 8086 & 8088 are later widely used in affordable personal computers. Some educators in the early 1980s foresaw the potential of these new technologies. Lin proposed using a microprocessor known as embedded control today to blend hardware and software education [3]. Pokoski pictured the future of digital system education consisting of "problem-solving, computer software principles, and computer hardware principles" as primary tools [4]. These past works indicated that computer programmers were expected to master hardware systems to code due to limited processing power and storage.

The education pattern shifted drastically in the next

2023 IEEE Integrated STEM Education Conference (ISEC) | 979-8-3503-0001-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISEC57711.2023.10402127







*Journal of Electronic Gaming and Esports*, 2023, 1, 1-12
https://doi.org/10.1123/jege.2022-0041


## The Internet Is Not Forever: Challe
## in Video Game Archiving a

**Brianna Dym,[1] Ellen Simpson,[2] Olivia**
[1]Roux Institute at Northeastern University, Portland, ME, USA; [2]University of Colorado Bou

Video games are an increasingly significant cultural touchstone in people'
video games faces unique challenges, including intellectual property law,
what it means to preserve a video game. In an exploratory study investigati
to 15 amateur game preservationists and hobbyists about their informal wo
play. We found a lack of access to particular games during childhood or yo
in other formats—such as emulated games they could play on other medi
computer). Their nostalgia and the communities they found searching for t
work. Participants leveraged distributed knowledge across their comm
interested in playing them. Considering these findings in the context of m
archive a game, especially when that game is dependent on interactive, c
current archival practices in contrast to informal, accidental archival wo

***Keywords:*** video games, digital archives, modding, romhacks

*I think with some games, Nintendo is very much like, oh, yeah, Mario World, we got ya. [ . . . ] But there's some games where it's either untranslated, like there's this game called Wagian Paradise. [ . . . ] And it's definitely not popular enough to warrant being put on Switch Online. Because like, who'd pay for that? But like, you know, I mean, I'd paid for that. —P15*

In 2023, Nintendo shuts down their eShop services for the 3DS and WiiU. This, along with Sony's announcement that they intended to shut down their online storefronts for the PlayStation 3 and PlayStation Vita, represents change for hobbyists who wish to play older games (Kuhnke, 2022). Although Sony eventually reversed their decision, the closure of digital marketplaces catering to older generations of consoles led to a loss of access for many gamers. In fact, as Oisin Kuhnke of *GameSpot* points out, with the closure of these virtual marketplaces, some games will be lost to players who wish to play the game legally and cannot afford to pay hundreds of dollars for a physical copy of the game cartridge (Kuhnke, 2022). Moreover, these physical cartridges will not last forever, and even the games preserved at official game archives, such as the Computer History Museum or the National Videogame Archive, run the risk of being lost due to the natural demagnetization of game storage on physical game cartridges and discs (Monnens, 2009). Archiving video games and consoles requires restoration work that is relatively new compared with preservation practices in mediums such as film or music (Kuhnke, 2022).

Video games exist in a complicated place in terms of archival work and cultural recognition—it is challenging to convince people who do not play games that they are an art form worth preserving (Kuhnke, 2022). Formal archival work is still determining what it can and cannot do in preserving video games as their code. The code itself is often proprietary and controlled by companies like

Dym (b.dym@northeastern.edu) is corresponding author.

Sony or
archival
(Newma
collectio
serving
about the
games—
hobbies
new gam
2013; N
compani
and mod
than any
Modified
Pokemon
(Frank, 2
received
notices.
videos o
As t
sustainab
selves bu
video ga
expand
working
Digital e
play gam
has mad
vists can
derivativ
their pra
practices
surround
In a
practices

74

## Evolution of Emacs Lisp

STEFAN MONNIER, Université de Montréal, Canada
MICHAEL SPERBER, Active Group GmbH, Germany

Shepherd: Brent Hailpern, IBM Research, USA

While Emacs proponents largely agree that it is the world's greatest text editor, it is almost as much a Lisp machine disguised as an editor. Indeed, one of its chief appeals is that it is *programmable* via its own programming language. Emacs Lisp is a Lisp in the classic tradition. In this article, we present the history of this language over its more than 30 years of evolution. Its core has remained remarkably stable since its inception in 1985, in large part to preserve compatibility with the many third-party packages providing a multitude of extensions. Still, Emacs Lisp has evolved and continues to do so.

Important aspects of Emacs Lisp have been shaped by concrete requirements of the editor it supports as well as implementation constraints. These requirements led to the choice of a Lisp dialect as Emacs's language in the first place, specifically its simplicity and dynamic nature: Loading additional Emacs packages or changing the ones in place occurs frequently, and having to restart the editor in order to re-compile or re-link the code would be unacceptable. Fulfilling this requirement in a more static language would have been difficult at best.

One of Lisp's chief characteristics is its malleability through its uniform syntax and the use of macros. This has allowed the language to evolve much more rapidly and substantively than the evolution of its core would suggest, by letting Emacs packages provide new surface syntax alongside new functions. In particular, Emacs Lisp can be customized to look much like Common Lisp, and additional packages provide multiple-dispatch object systems, legible regular expressions, programmable pattern-matching constructs, generalized variables, and more. Still, the core has also evolved, albeit slowly. Most notably, it acquired support for lexical scoping.

The timeline of Emacs Lisp development is closely tied to the projects and people who have shaped it over the years: We document Emacs Lisp history through its predecessors, Mocklisp and MacLisp, its early development up to the "Emacs schism" and the fork of Lucid Emacs, the development of XEmacs, and the subsequent rennaissance of Emacs development.

CCS Concepts: • **Social and professional topics → History of programming languages**.

Additional Key Words and Phrases: History of programming languages, Lisp, Emacs Lisp

**ACM Reference Format:**
Stefan Monnier and Michael Sperber. 2020. Evolution of Emacs Lisp. *Proc. ACM Program. Lang.* 4, HOPL, Article 74 (June 2020), 55 pages. https://doi.org/10.1145/3386324

Authors' addresses: Stefan Monnier, Université de Montréal, C.P. 6128, succ. centre-ville, Montréal, QC, H3C 3J7, Canada, monnier@iro.umontreal.ca; Michael Sperber, Active Group GmbH, Hechinger Str. 12/1, Tübingen, Germany, sperber@deinprogramm.de.


2475-1421/2020/6-ART74
https://doi.org/10.1145/3386324

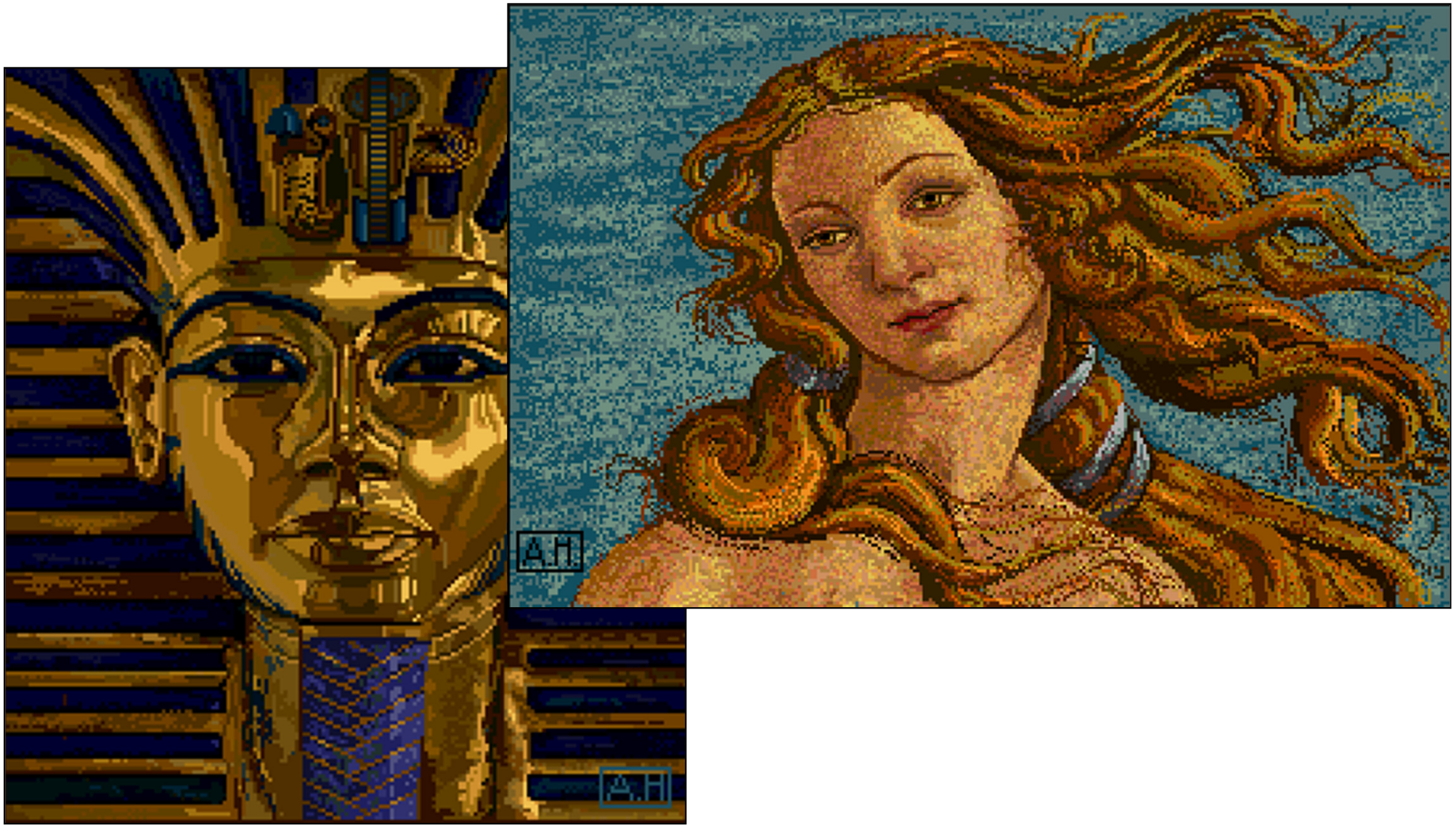
# Gallery

https://amiga.lychesis.net/
https://amiga.abime.net/games/view/the-secret-of-monkey-island#screenshots

# Gallery

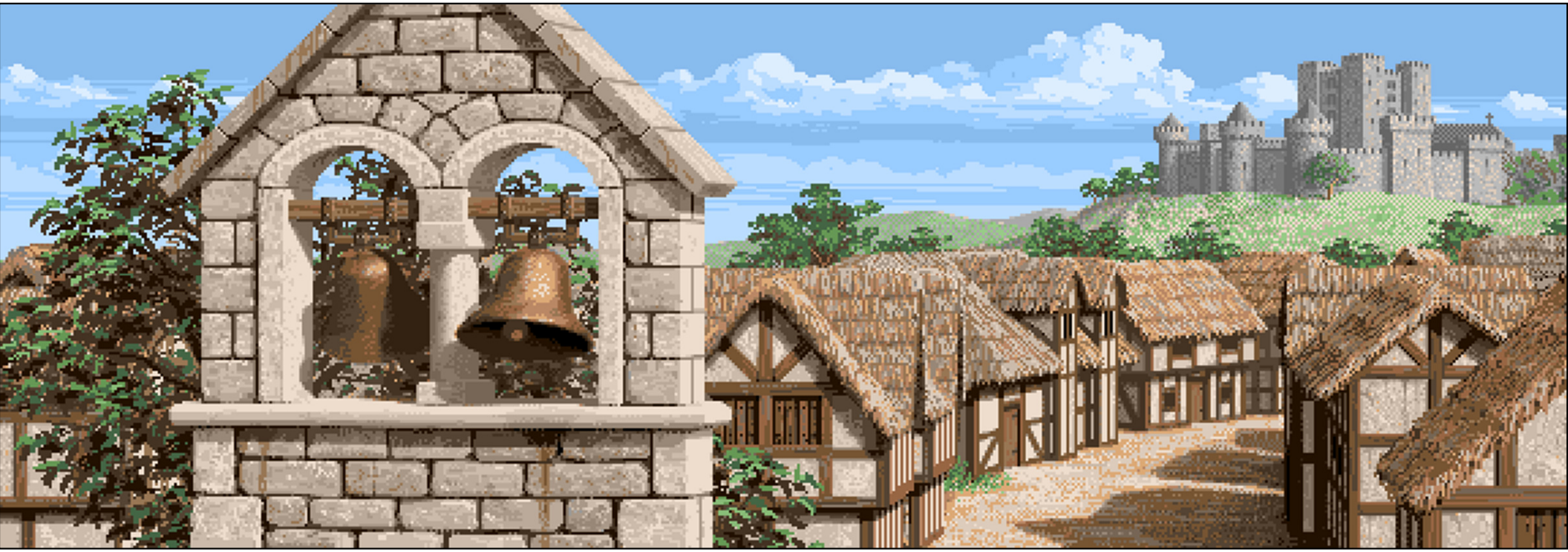
https://amiga.lychesis.net/
https://amiga.abime.net/games/view/the-secret-of-monkey-island#screenshots

# Gallery




https://amiga.lychesis.net/
https://amiga.abime.net/games/view/the-secret-of-monkey-island#screenshots

# Gallery



https://amiga.lychesis.net/
https://amiga.abime.net/games/view/the-secret-of-monkey-island#screenshots


He sure looks nice exploding
against the night sky.

# Conclusion

What are the constraints that guided the architecture, design, and implementation of Deluxe Paint to handle the computational and memory constraints of the Amiga 1000?

What are the constraints that guided the architecture, design, and implementation of Deluxe Paint to handle the computational and memory constraints of the Amiga 1000?

7/38

- RQ1: How and by whom was DPaint developed, and what was its overall quality?
- RQ2: What architectural styles can be observed in the source code of DPaint?
- RQ3: What design patterns are present in the source code of DPaint?
- RQ4: What coding idioms are present in the source code of DPaint?
- RQ5: How was complexity distributed across the different components of DPaint, and why?

9/38

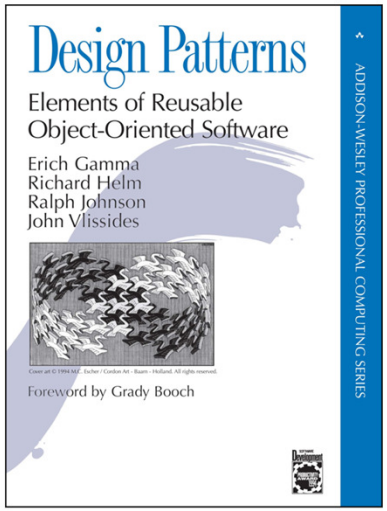
What are the constraints that guided the architecture, design, and implementation of Deluxe Paint to handle the computational and memory constraints of the Amiga 1000?

7/38

- RQ1: How and by whom was DPaint developed, and what was its overall quality?
- RQ2: What architectural styles can be observed in the source code of DPaint?
- RQ3: What design patterns are present in the source code of DPaint?
- RQ4: What coding idioms are present in the source code of DPaint?
- RQ5: How was complexity distributed across the different components of DPaint, and why?

9/38

# Discussions




"DPaint's developers independently arrived at structural solutions that the GoF would formalise nine years later, using function pointers, struct embedding, and dispatch tables in place of classes and virtual methods."

30/38

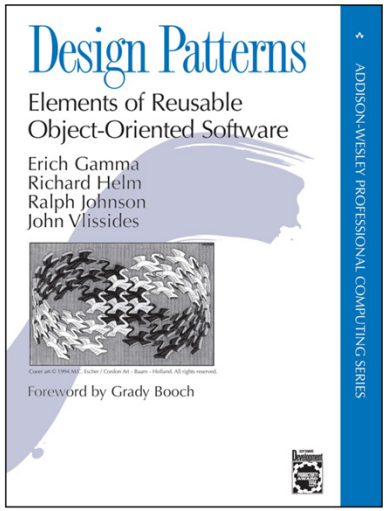
What are the constraints that guided the architecture, design, and implementation of Deluxe Paint to handle the computational and memory constraints of the Amiga 1000?



- RQ1: How and by whom was DPaint developed, and what was its overall quality?
- RQ2: What architectural styles can be observed in the source code of DPaint?
- RQ3: What design patterns are present in the source code of DPaint?
- RQ4: What coding idioms are present in the source code of DPaint?
- RQ5: How was complexity distributed across the different components of DPaint, and why?



# Discussions




"DPaint's developers independently arrived at structural solutions that the GoF would formalise nine years later, using function pointers, struct embedding, and dispatch tables in place of classes and virtual methods."



# Discussions

"Without version control, IDEs, linters, or automated testing, DPaint source code compiles with only warnings on a modern compiler and runs correctly on original hardware. The code's quality is evidenced by its commercial success and […] compilability 40 years later."

https://www.classic-computers.org.nz/collection/ibm-at.htm