Enriching Legacy Systems

A field report about the architectural work on performance critical and long-living software.

Dr. Robert Sorschag

ECSA 2014, Vienna

Industry Day

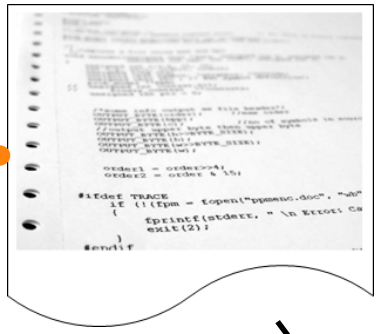
Enriching Legacy Systems

New Features

Improve Quality

Source Code

Challenges



Techniques and Tools

Platforms

Embedded Devices

Contents

- **Zühlke**
  - Company profile
- **Project**
  - Costumer
  - Requirements
  - Architecture
  - Demo
- **Architectural Work**
  - New features
  - Improved code quality
  - Techniques and tools
- **Conclusions**

ZÜHLKE

Empowering Ideas

Zühlke Engineering
History

1968: Founded by Gerhard Zühlke

- Switzerland, Zurich
- Product Innovation
- Medical Products

1973: Software Engineering

1980: Management Consulting

1998: Germany

2001: UK

2009: Austria

2013: Serbia

2014: ~650 Employees

Zühlke Engineering
Locations

1968: Founded by Gerhard Zühlke

- Switzerland, Zurich
- Product Innovation
- Medical Products

1973: Software Engineering

1980: Management Consulting

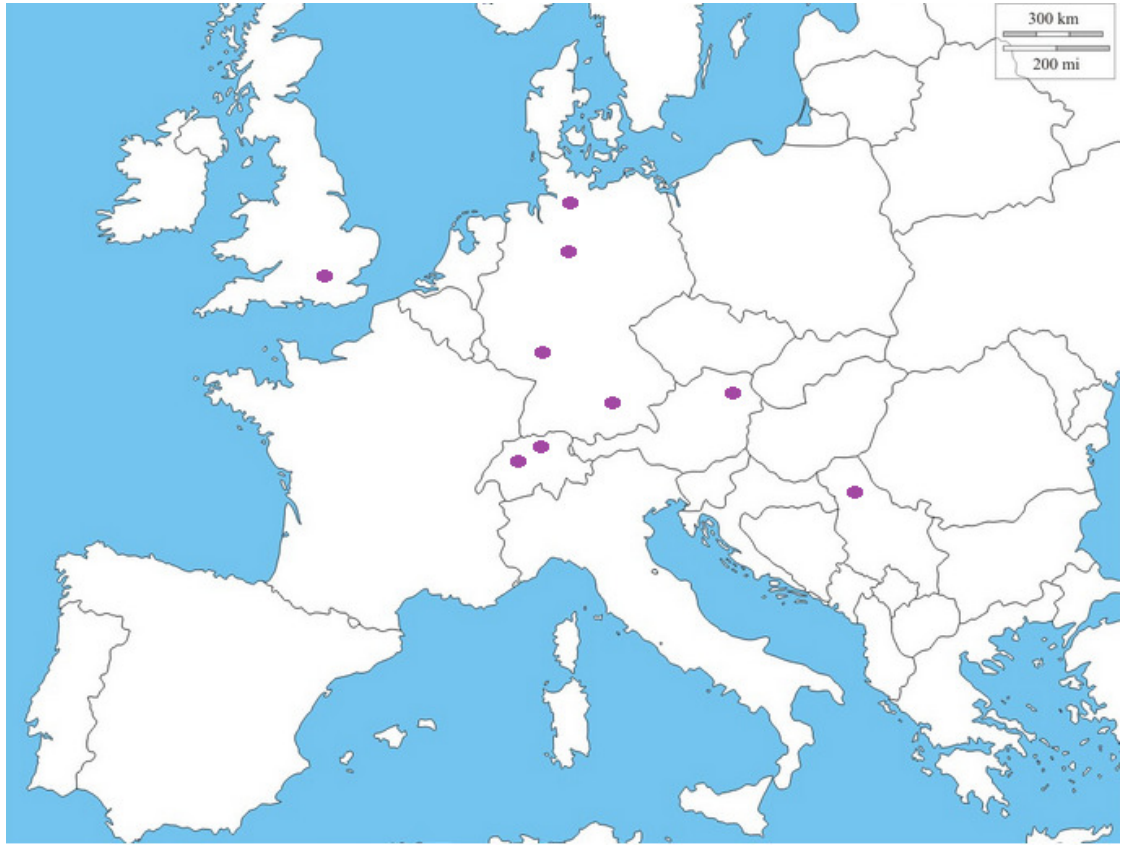
1998: Germany

2001: UK

2009: Austria

2013: Serbia

2014: ~650 Employees

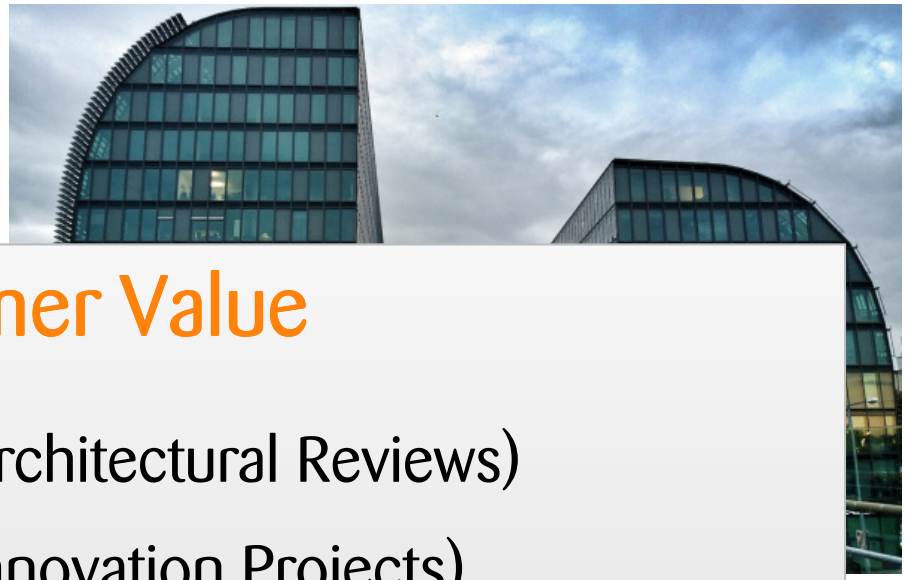


Zühlke Engineering
Austria

- **Office in Vienna**
  - Rivergate
  - Handelskai 19, 1200 Vienna
- **17 Employees**
- **Sectors**
  - Machine-to-machine solutions
  - Security-critical system development
  - Business-critical applications
  - Software architecture
  - Requirements engineering
  - Testing
  - Product innovation

Zühlke Engineering
Austria

- **Office in Vienna**
  - Rivergate
  - Handelskai 19, 1200 Vienna
- **17**
- **Se**
  -
  -
  -
  - Software architecture, requirements engineering, testing
  - Product innovation



Customer Value

- Verification of design (Architectural Reviews)
- Missing know-how (Innovation Projects)
- Missing resources (Implementation Projects)

Project

Enriching Legacy Systems

Customer
Manufacturer of Construction Machines

- **Construction Machines**
  - Building Material Machines
  - Mining Machines
  - Construction Vehicles
  - Cranes
- **Displays**
  - Visualization with touch screen
  - Machine data and navigation
  - Rear view camera
  - Similar displays in all machines
  - HW & SW in-house development

Displays
Hardware

- **Development**
  - Started in early 2000's
    - 1st Generation (2003)
    - 4th Generation (2013)
  - Touch screen
- **Used under Extreme Conditions**
  - Dirt, sand, snow, ice
  - Changing lighting - direct sunlight and night operation
- **Long-lasting Product**
  - Warranty of 20 years
  - Almost indestructible
    - -40° to +70° C
    - Water-, dust- and shock-proof

CPU: Single-core ~200 MHz PPC

Screen: 7", 800 x 480 pixel

CPU: Quad-core ~1000 MHz ARM

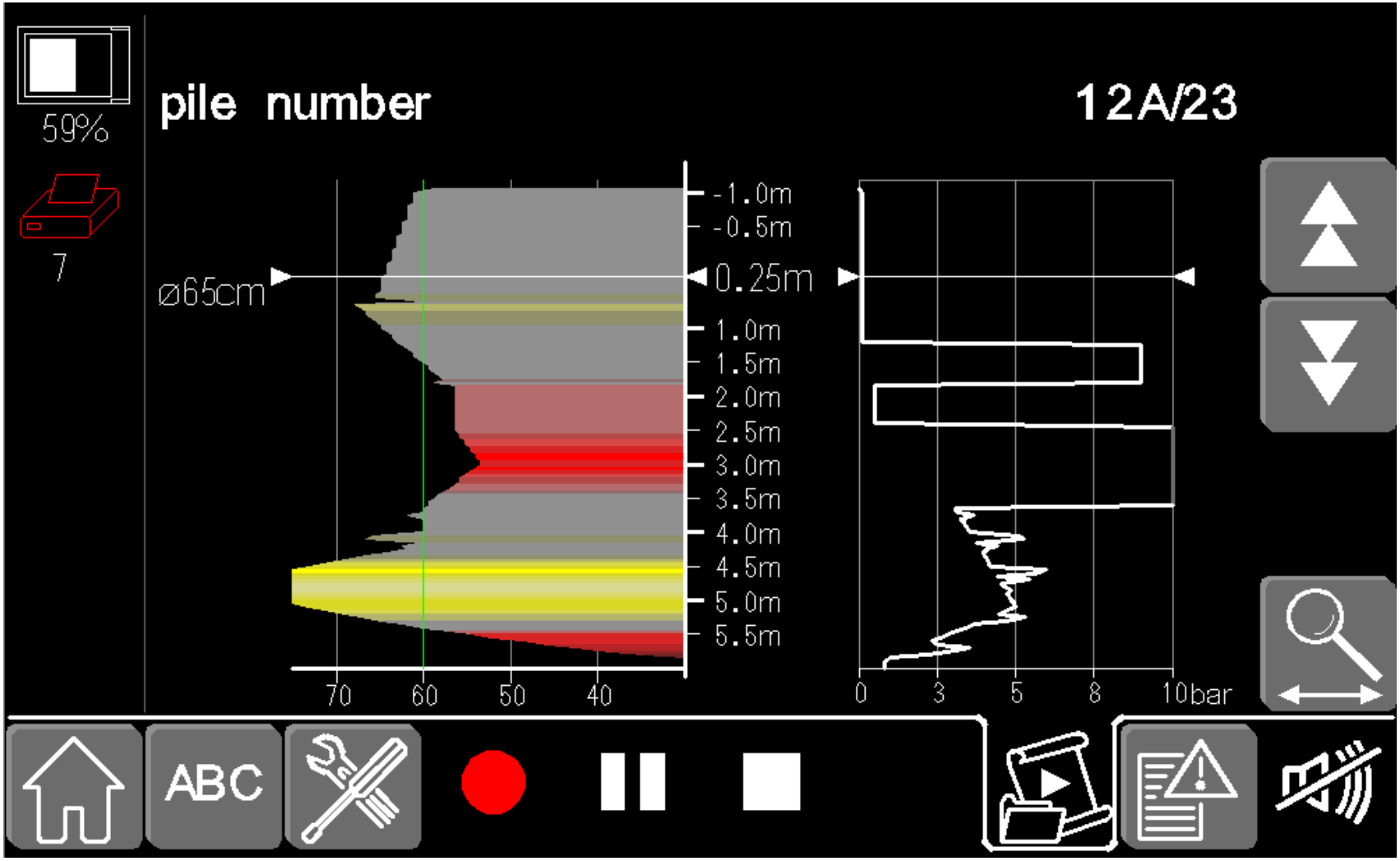
Screen: 10,4", 1280 x 960 pixel

Displays
Software

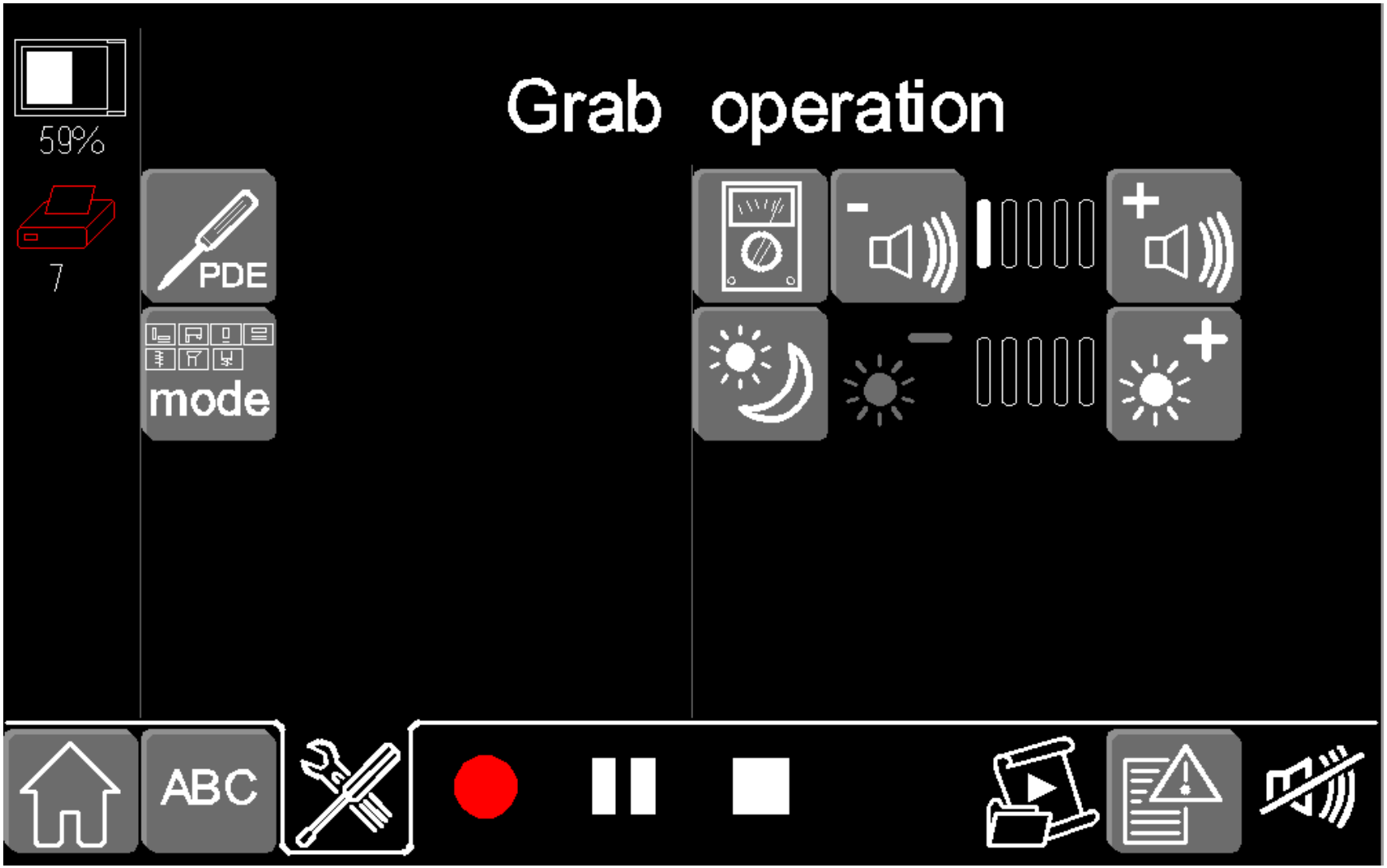
zühlke
empowering ideas

- **Development**
  - C / C++
  - Real time OS (VxWorks)
- **Requirements**
  - Safety, security and performance critical
  - Update
  - Remote Maintenance
  - Backwards compatibility
- **Graphic is Out-dated**
  - Vector-based rendering engine (integer arithmetic)
  - No anti-aliasing and sub-pixel rendering
  - No color gradients
  - No font type support (e.g. True Type Fonts)

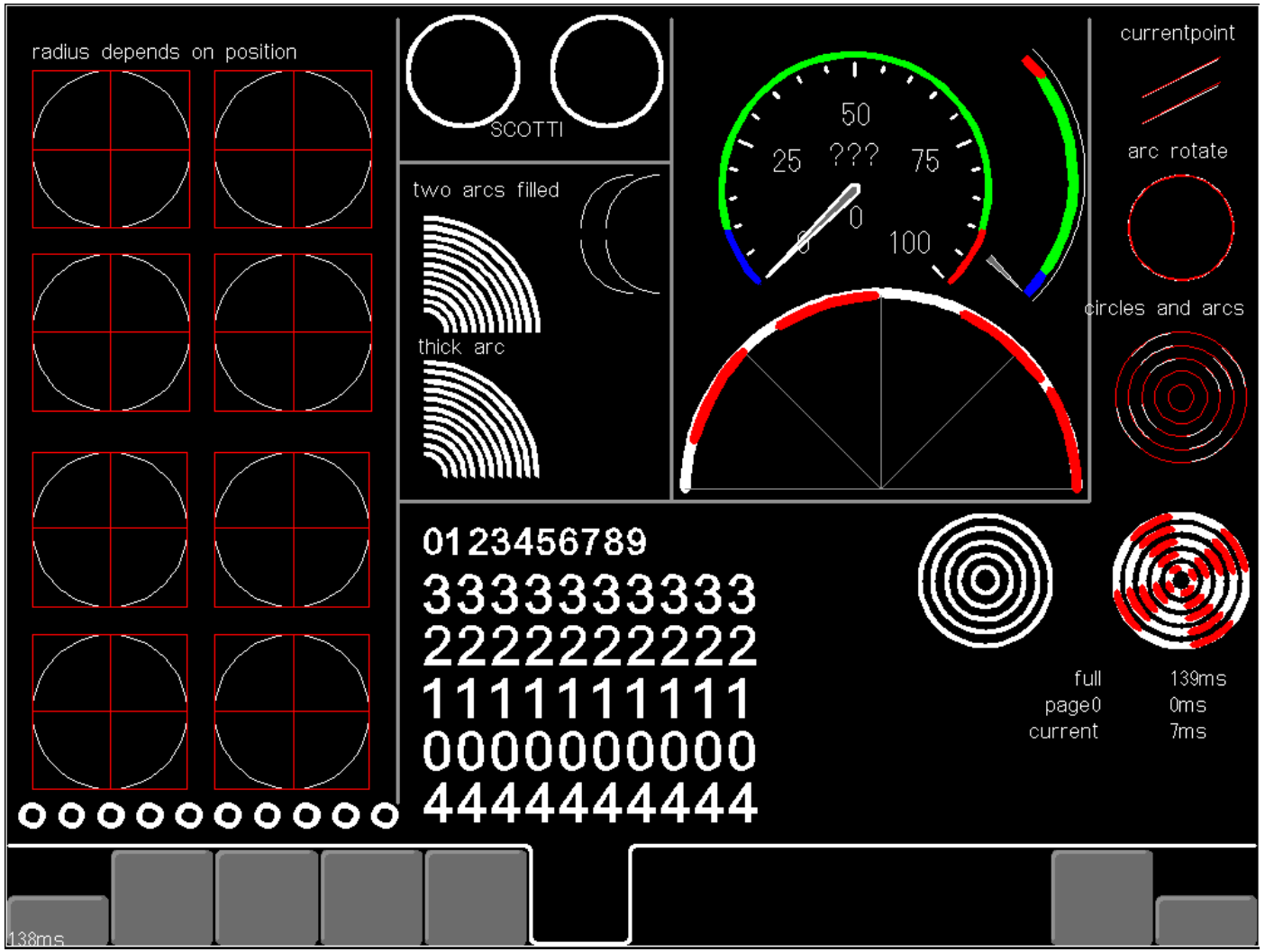
Example
Drill Navigation



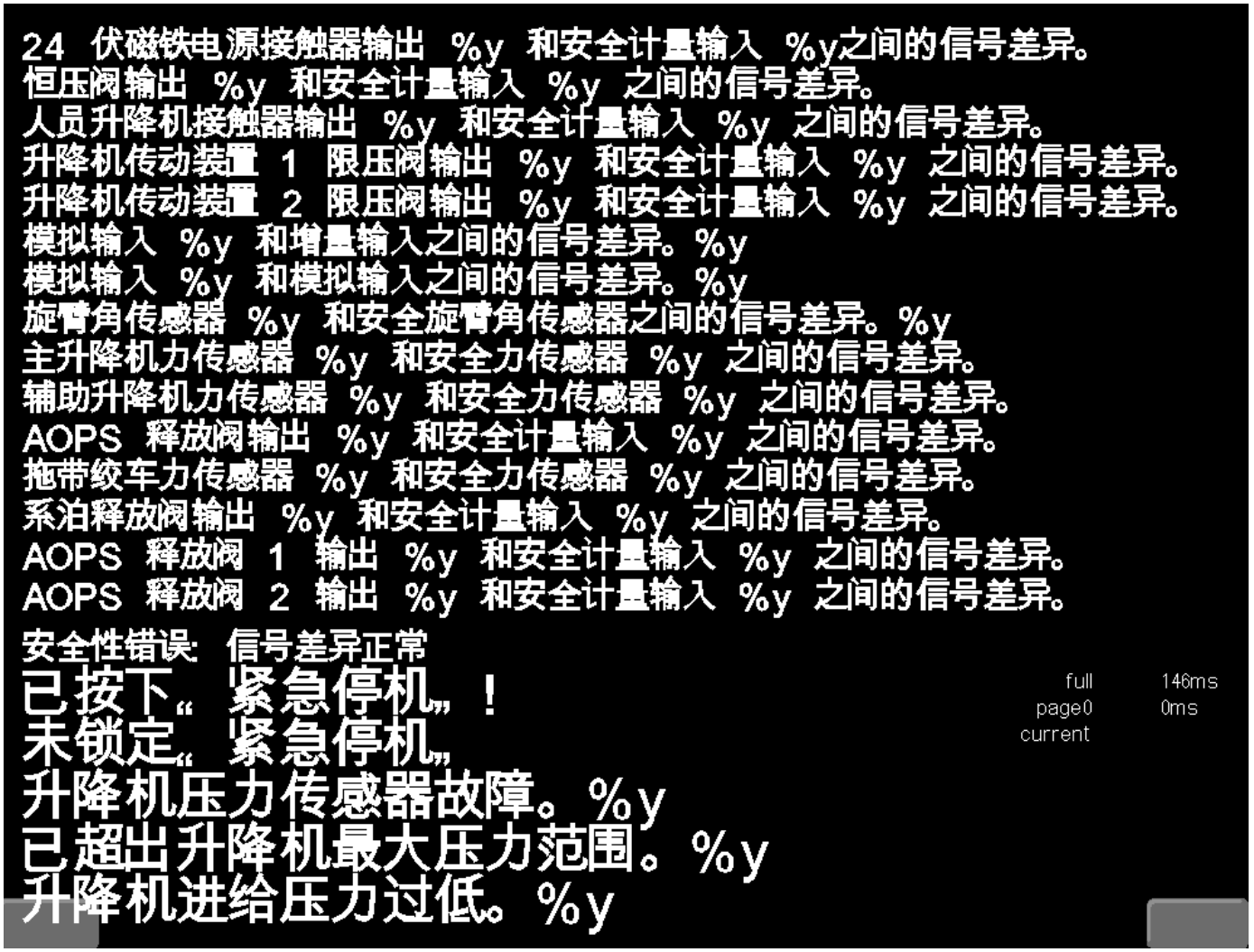
Example
Display Configuration



Example
Circles and Arcs



Example
Chinese fonts



Project
Enriching Legacy Systems

Goal

- The graphic should look nice and modern
- Integrate a new rendering engine

Architecture
Display Software

- **Visualizations**
  - Written in scripting language
    - Drawing operators (e.g. moveto, lineto)
    - Similar to PDF and PostScript
  - Visual object libraries (e.g. tachometer)
  - Graphical UI designer
- **Compiler**
  - Compiles visualization script into intermediate byte-code
- **Byte-code Interpreter**
  - Executes byte-code
  - Renders visualizations
  - Communicates with machine sensors

```
%?page
%?fn     OpticMain
%?brief Page with a single tachometer
%?bbox   0 0 960 720
/OpticMain {
    matrix currentmatrix
    currentcolor
    650 580 translate
    0 rotate
    1.5 1.5 scale
    0 false (???) 0 100 10 90 1 2 4 1 true
    baRoundDisplay001
    setcolor
    setmatrix
} def
```

Architecture
Display Software

- **Visualizations**
  - Written in scripting language
    - Drawing operators (e.g. moveto, lineto)
    - Similar to PDF and PostScript
- **Compiler**
  - Translates visualizations into intermediate byte-code
- **Byte-code Interpreter**
  - Executes byte-code
  - Renders visualizations
  - Communicates with machine sensors

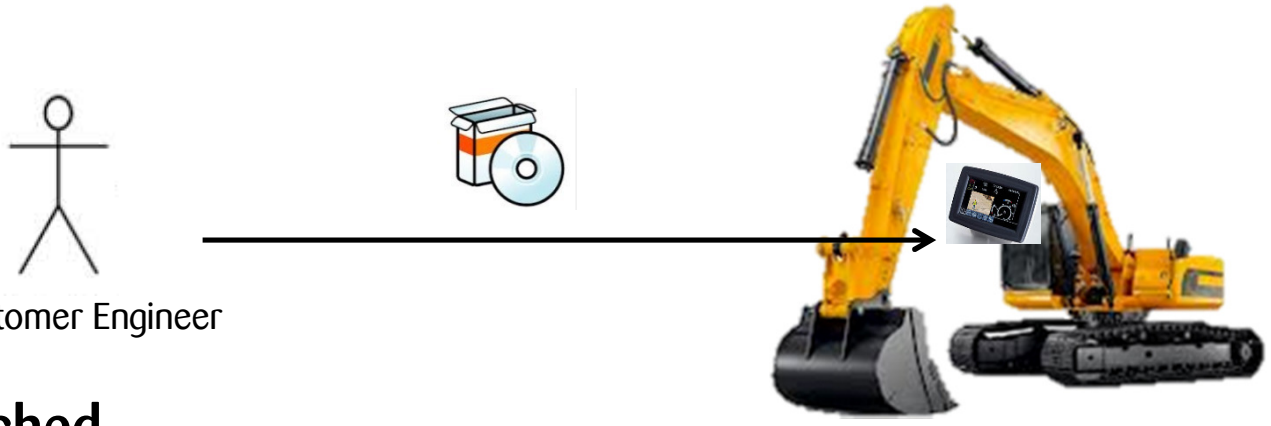
```
%?page
%?fn     OpticMain
%?brief Page with a single tachometer
%?bbox   0 0 960 720
/OpticMain {
    matrix currentmatrix
    currentcolor
```

true

Challenge

Warranty of 20 years

Use Case
Update Display Software



- **Untouched**
  - Display (HW)
  - Visualization script
- **New**
  - Display Software
    - Compiler
    - Byte-code Interpreter + rendering engine
    - OS kernel
  - Compatible to display generation

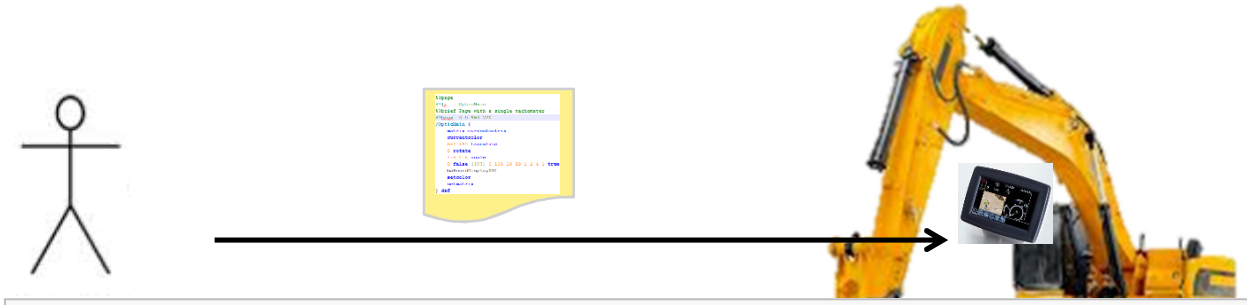
- **Process**
  - Upload display software
  - Restart display
    - Compile visualization
    - Store in permanent storage
  - Start visualization

Use Case
Update Visualization

- **Untouched**
  - Display (HW)
  - Display Software
- **New**
  - Visualization script
  - Compatible to compiler
    - New compiler might support new features
    - E.g. color gradients

- **Process**
  - Upload visualization
  - Restart display
    - Compile visualization
    - Store in permanent storage
  - Start visualization

Use Case
Update Visualization



Cus

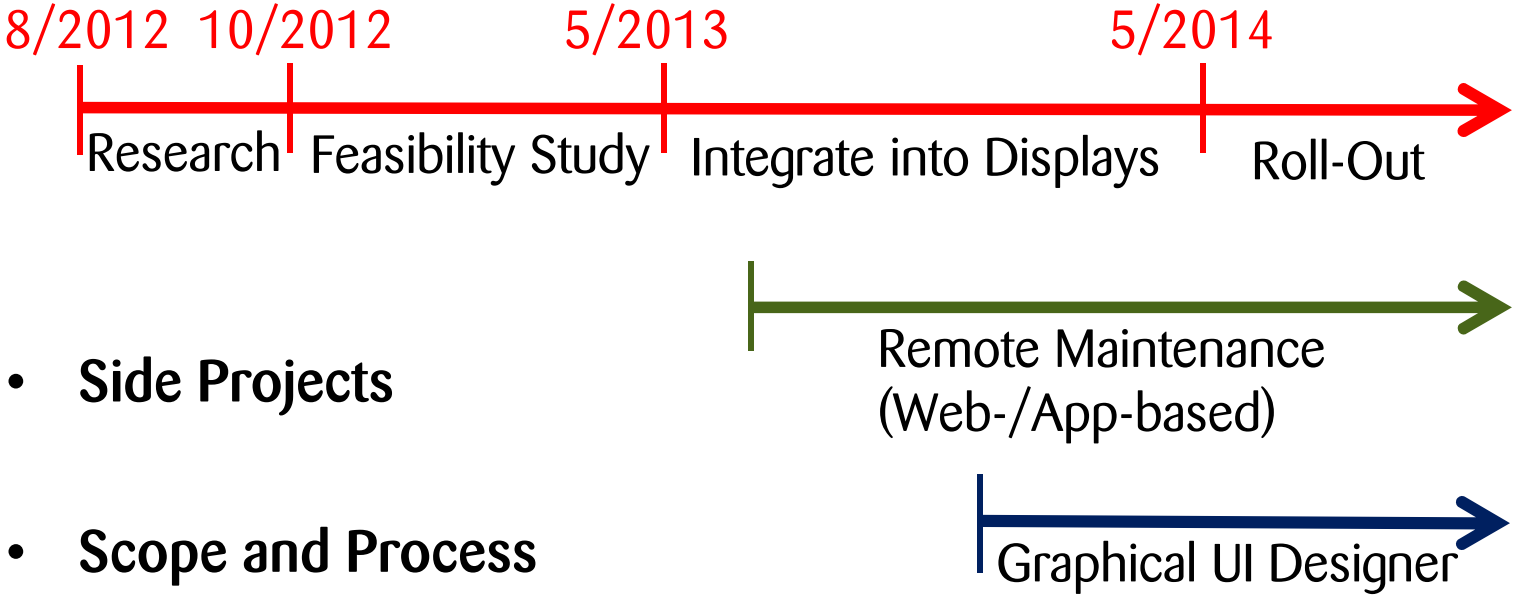
- **Old**
  - Disp
  - Disp
- **New**
  - Visualization script
  - Compatible to compiler
    - New compiler might support new features
    - E.g. color gradients

Challenges

- Security
- Remote maintenance

- Restart display
  - Compile visualization
  - Store in permanent storage
- Start visualization

Project
Process and Timeline



- **Side Projects**
- **Scope and Process**
  - Iterative development
  - Scrum-like process (3 week sprints)
  - 2 + ½ persons
  - 2 years

Research
Select Rendering Engine

zühlke
empowering ideas

- **Cairo**
  - 2-D graphics rendering library
  - Support multiple outputs
    - Different frame buffers: RGB, RGBA, RGB565
    - Vector graphics: SVG, PDF
    - Compressed images: PNG, JPG
    - Experimental: OpenGL, DirectFB
  - Modern graphic rendering
  - Part of the GIMP Tool Kid (GTK)
  - Open-Source (LGPL and MPL)
- **API**
  - Interface similar PostScript and PDF
  - Pure C
  - All methods use context data to simulate object orientation

cairographics.org

Feasibility Study
Prototype Development

- **Prototype**
  - Integrate Cairo to display software
  - Windows (Graphical UI designer)
  - Change legacy code only where necessary
  - Basic separation of modules and layers
    - Interface between byte-code interpreter and rendering engine
- **Functionality and Performance**
  - Rendering quality
  - Performance on host (Windows)
- **Estimation about Productive Usage**
  - Architectural changes
  - Required performance improvements

Integrate into Displays

zühlke
empowering ideas

- **Roadmap**
  - Start with best hardware (G4 displays)
  - Internal test user (visualization designer)
  - Field test with one construction machine type
- **New Features**
  - New rendering engine
    - Greenfield approach (don't care about old rendering engine first)
    - Class hierarchy for different backends
    - Fallback mode (old rendering engine)
  - Runtime switch between rendering engines
  - Language support
  - Color gradients
    - Changes of in visualization script compiler
  - Multi-platform, multi-threading and multi-instance
  - Improved accuracy (floating point vs. integer arithmetic)

Example
Color Gradients on Display

zühlke
empowering ideas

Demo

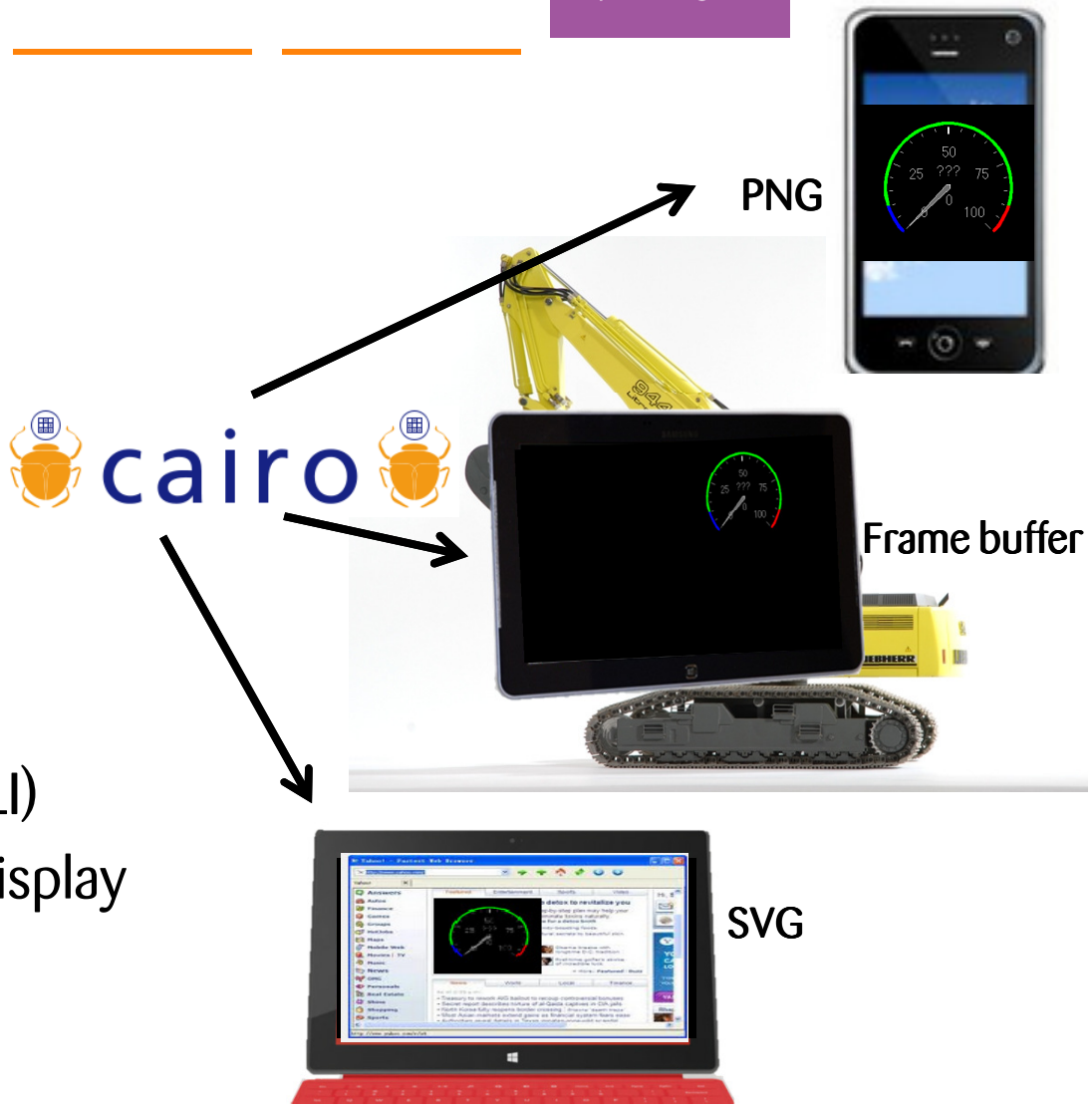
Improved Graphic Rendering

Side Projects

zühlke
empowering ideas

- **Remote Maintenance**
  - Web-based services
  - App-based services
  - Port to ARM and Android

- **Graphical UI Designer**
  - Display software
  - C++ to C# Interface (C++/CLI)
  - Generate visualizations for display



Architectural Work

Enriching Legacy Systems

Code Quality

- **Legacy Code**
  - High complexity
  - Many configurations and versions
  - Many sub-modules
  - Different development teams
  - Design and coding style
  - Code smells (e.g. copy & paste)
  - Cyclic dependencies
- **Customer View**
  - Feature driven development
  - Replacement of existing parts only if absolutely required
  - ~5% of project costs to improve code quality
  - Operative business

Code Quality

- Le
- Cu

Techniques

- Boy scouts approach
  - Step by step improvements
- Automated Tests (on host and target)
  - Smoke tests
  - Unit tests
  - Integration tests
- Deployment
  - Continuous integration (Jenkins, Cruse Control)
  - Application Lifecycle Tools (Jira)
- Version Control
  - SVN and configuration tool

Portability

- **Processor Architectures**
  - x86
  - ARM
  - PPC
- **Operating Systems**
  - VxWorks (6.3, 6.7)
  - Windows
  - Android
  - Embedded Linux
- **Compiler and Language Bindings**
  - Gcc, Visual Studio (vc11, vc12)
  - C++ Standards (C++11) and standard libraries (boost)
  - C#
  - Python

Portability

- P
  -
  -
  -
- C
  -
  -
  -
  -
- C
  -
  -
  -
  - Python

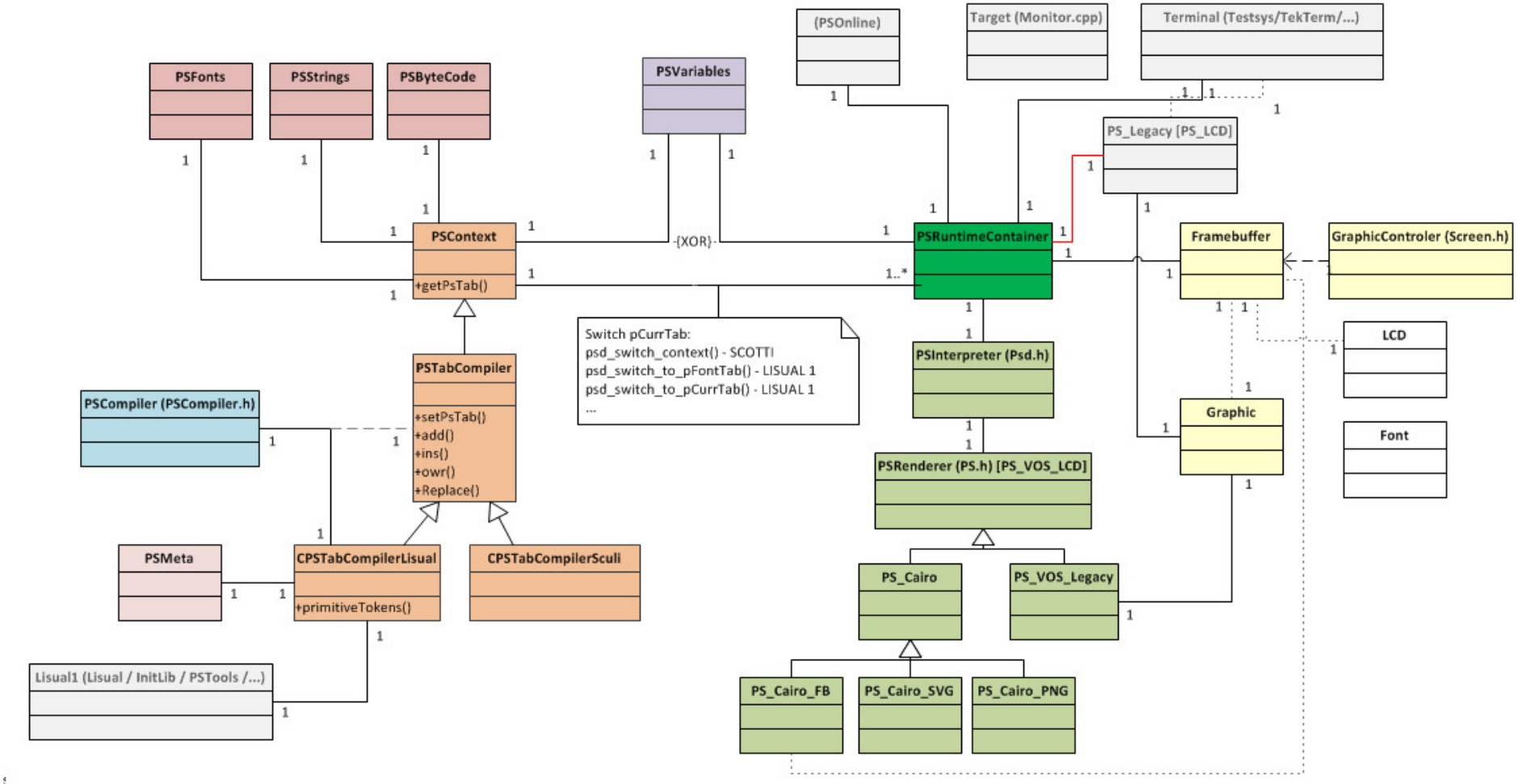
Techniques

- Specified data types
  - typedef signed long sint32_t;
- Hardware Abstraction Layer
  - #ifdef ARM / PPC / x86
  - In one module
- Develop at most comfortable environment possible
  - Debugging, auto-completion and many more
- Build-Server for target compilation (in all configurations)
- Static code analysis
- CLR Interface

API Design
CLR Interface for C++ to C#

- **Intermediate language**
  - Windows and mono port
  - Wrapper classes with data type conversion
- **API**
  - Use same code as displays
  - Multiple instances
    - One visualization compiler
    - Many byte-code interpreter and rendering engines
  - Object oriented design
    - Redesign of entire display software
    - Everything a class now
  - Separation between read and write access

API Design
CLR Interface for C++ to C#



Long-Living Software

- **Warranty of 20 years**
  - Support for Hardware and Software
  - Updates
  - Compatibility between Displays – Software – Visualization
- **Safety critical**
  - Run at least 24h
  - At least 8 fps
  - Restart after 1 sec freeze
- **Bugs reported in field are painful**
  - Ok for new features
  - Old features should not fail

Long-Living Software

- W
  -
  -
  -
- Sa
  -
  -
  -
- Bu
  -
  -

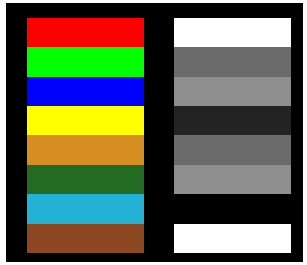
Techniques

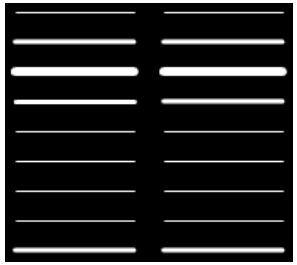
- Agile processes
  - Continuous releases
  - Iterative design and development
- More automated tests
  - Stress tests
  - **Visual tests**
  - Memory leak detection
  - Back-to-back tests (for code replacement)
  - Static code analysis
  - Reviews (4 eyes principle)

Long-Living Software
Visual tests

zühlke
empowering ideas

- **Standard Approach**
  - Pixel comparison not sufficient (anti-aliasing)
  - Existing tests suits do not work on target
- **Probabilistic Measures**
  - % of black/white/red/… pixels
  - % identical pixels
  - Color distance instead of pixel comparison
- **Good test images**





Performance Critical

- **Requirements**
  - At least 8 fps
  - Responsive (new frame after user interaction)
  - Data exchange with machine control
- **Computation**
  - Cairo uses double arithmetic
  - Old graphic uses integer arithmetic
  - First display generations have no FPU

Performance Critical

- Re
  -
  -
  -
- Co
  -
  -
  -

Techniques

- Measure first
  - Use productive environment (important: Release)
  - Find bottlenecks
  - Utilize code (CPU ticks)
  - Profiler
- Special Case for Bottlenecks
  - Hardware Abstraction Layer (e.g. use intrinsic)
  - **Frame rendering**
  - Switch between two views

Performance Critical
Frame Rendering

- **Cairo**
  - Supports clipping
  - Only clipped regions are newly rendered
- **Region of Interest**
  - Store byte-code instruction of consecutive frames
  - Compare calls
  - Find different calls
  - Compute rectangles and combine them
- **Architecture**
  - Independent from rendering backend
  - Cairo for real-time calculations (e.g. currentpoint)

Conclusions

Enriching Legacy Systems

Conclusions

- **Work on Legacy Software**
  - Challenging but possible
  - Complexity grows over time
- **Add new features and to improve quality**
  - Start locally
  - Continue step-by-step
  - Agile process
  - Domain knowledge is important
  - Automate tests wherever possible
  - Tool support

Enriching Legacy Systems

A field report about the architectural work on performance critical and long-living software.

Dr. Robert Sorschag

ECSA 2014, Vienna

Industry Day