

Automating Software Development with UML 2.0

Cris Kobryn

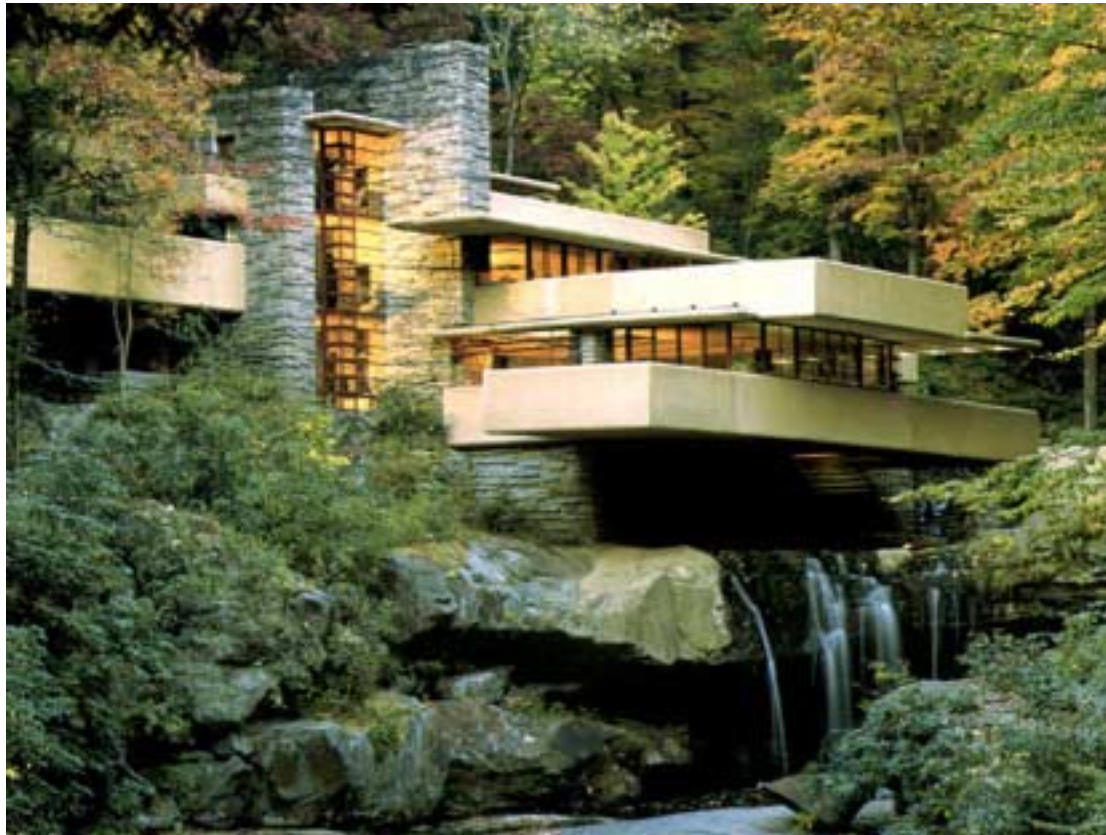
Chief Technologist

cris.kobryn@telelogic.com

Overview

- Problems
 - the software paradox revisited
- Analysis
 - what have we learned?
- Solutions
 - how can we make users more successful?
 - customer experiences

Conceptual Architecture: Fallingwater Syndrome



Fallingwater:

<http://www.adelaide.net.au/~jpolias/FLW/Images/FallingWater.jpeg>

Technical Architecture: Tower-of-Pisa Syndrome



Tower of Pisa, West View:

<http://www.endex.com/gf/buildings/ltpisa/ltpgallery/freepix/ltpgalleryfree.htm>

Technical Architecture: Ball-of-Mud Syndrome



<http://www.macalester.edu/~jschatz/residential.html>

What Have We Learned?

- Consider
 - “We learn more from buildings that fall down than from buildings that stand up.”
 - *What We Learned about Tall Buildings from the WTC Collapse*
Discover, Oct. 2002
 - we learn more from experience than theory
- Is there a relevant software analogy?
 - should we start answering hard questions ...
 - ... or should we continue to make the usual excuses that software industry is nascent, different, etc.?
- What have learned from software buildings?

Lessons Learned

- Software development remains human-intensive, unpredictable and error-prone
 - a cottage industry of skilled craftsmen rather than a mature industry that applies engineering and automation techniques
 - far behind our hardware engineering sibling, which boasts Moore's Law
- Software architectures are not inherently different from hardware and building architectures
 - some differences apply, since software is unusually malleable and (for most stakeholders) tends to be invisible
 - software is more amenable to engineering and automation than most practitioners care to admit!

Lessons Learned (cont'd)

- How do we build software systems “better, faster, cheaper”?
- Problem is not a simplistic time vs. quality paradox
 - expert, prolific programmers can accelerate quality solutions, but are scarce
 - compounded by cultural tensions – some developers favor more engineering and automation, while others prefer status quo
 - more of a Gordian knot, than a simplistic dialectic

Software Development Paradox

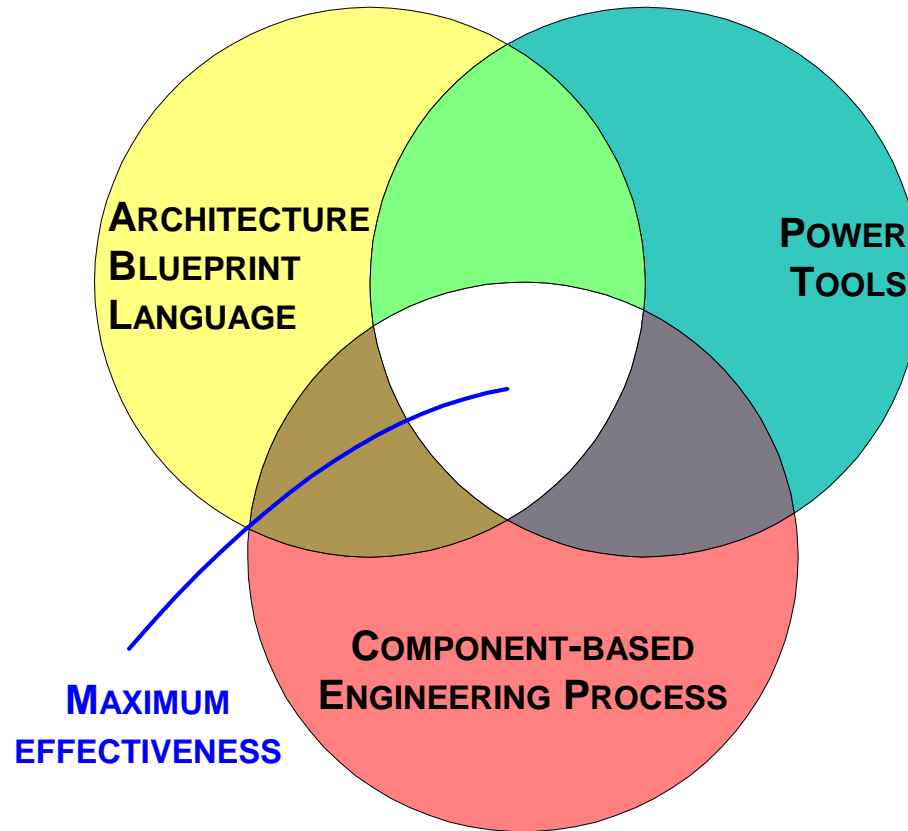


http://www.maa.org/devlin/devlin_9_01.html

Urgency

- We must resolve the software development paradox posthaste or suffer consequences
 - “By 2007, 70% of the world's computer programming activity will be in developing countries. Companies such as Accenture, EDS and IBM will go out of business unless they adapt by farming out their programming to countries such as India and China and concentrating on high-value project management.”
 - » George Colony, Forrester Research CEO
Computer Weekly, 16 May 2002
- However, no silver bullet (sword cut) solution appears imminent

Solution Triad



Solution Triad

- Architecture blueprint language
 - common language across software lifecycle for multiple stakeholders
 - sufficiently precise to automate generation of production-quality code
 - capable of supporting model-driven development (compare broken round-trips)
 - customizable for domains (telecom, auto, mil/aero) and platforms (J2EE, .NET)
- Power tools
 - use the blueprint language to automate code generation from specifications
 - amplify productivity of better-than-average developers
 - free developers from implementation minutiae so they can architect, analyze, design and test
- Engineering process
 - component-based rather than object-oriented
 - medium-weight and customizable (cf. lightweight and trivial, or robust and onerous)

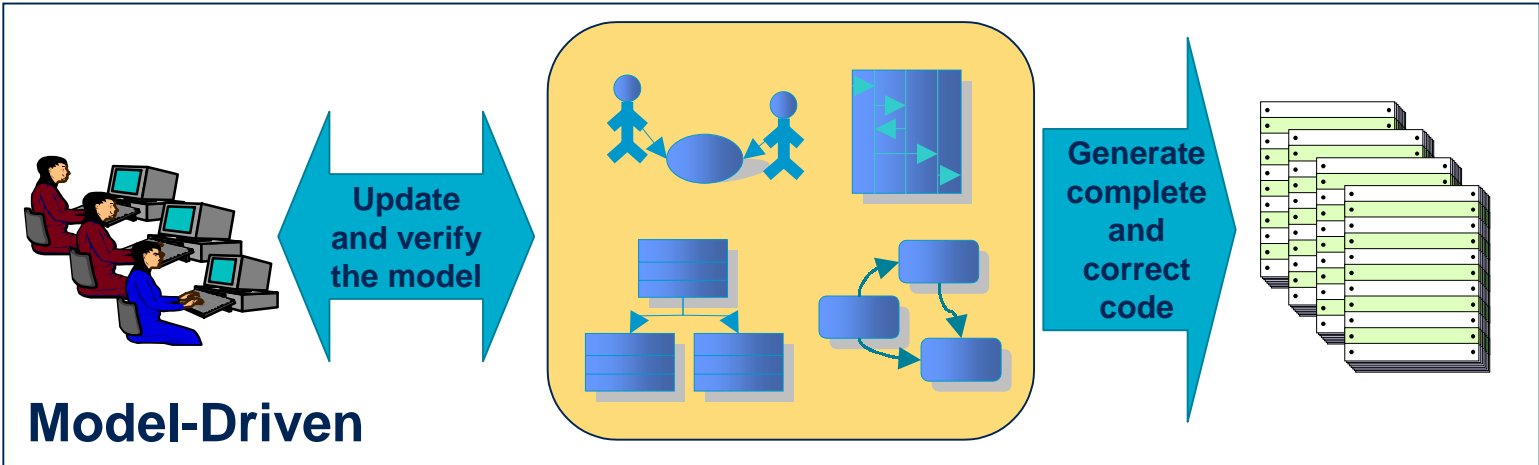
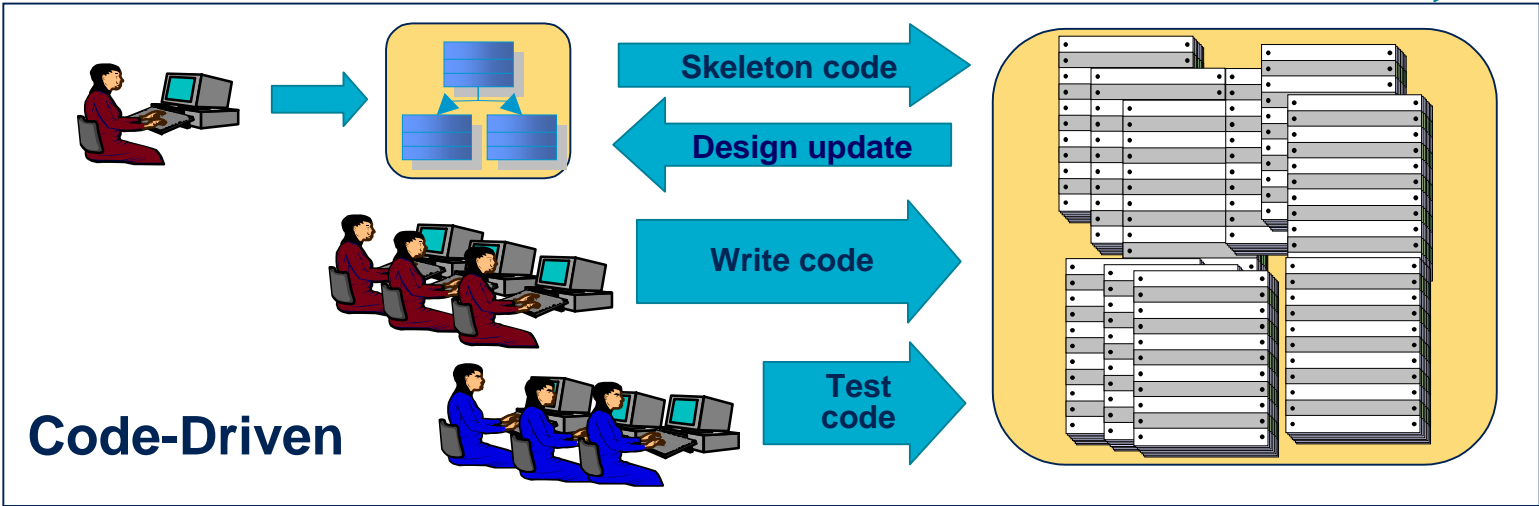
Telelogic Solution

- Architecture blueprint language: UML 2.0
 - evolving UML 2.0 standard promises to be a mature specification language (compare SDL, MSCs)
- Power tools: Tau/Developer & Tau/Architect
 - based on UML standard, incorporate best ideas proposed for UML 2.0
 - automates code generation and system verification
- Engineering process: BYOP
 - UML 2.0 and Tau/Developer can be flexibly used with any agile or robust engineering process
 - adapt methods to meet customers' evolving requirements

UML 2.0 as Enabling Technology

- UML 2.0 marks a major milestone in model-driven development
 - significantly more mature than UML 1.x
 - more precise, concise and consistent
- Major improvements include enhanced support for
 - component-based development
 - architectural specifications of large, complex systems
 - more scaleable, precise and integrated behavioral models
 - executable models
 - systems engineering

Code-Driven vs. Model-Driven



Customer Experiences: Tau Generation1 Tools (SDL-based)

- "Using Telelogic Tau led to 53% time savings, and helped us achieve platform independence of the product."
 - **Vocalis, UK**
- "In the future, we estimate that productivity will double during the design phase as Telelogic Tau greatly shortens the development time."
 - **Fujitsu, Japan**
- "One thing is clear, it enabled us to develop the [products] much faster than with traditional programming."
 - **Nokia, Finland**
- "The reliability of the generated code is very good."
 - **Lucent, UK**
- "We have not found a single error from the automatic code generation."
 - **Motorola, India**

Customer Experiences: Tau Generation1 Tools (SDL-based)

- "Our development time has been cut with 50%."
– **Ericsson Erisoft, Sweden**
- "Lead-time was cut with 25%, development cost with 27%, and there have been no faults detected so far."
– **Ericsson Ltd, UK**

Case Study: NEC

- Example of Automated Application Generation using Tau Generation1 technology
- Project: protocol stack for mobile phone
 - Program size: 40k lines of C
 - Productivity: 110% (slightly better than hand coding)
 - Memory footprint: 130% (compared to hand-crafted code but still within planned size)
 - 97% of errors were found before integration testing
- ***Development was finished in 4 months, as scheduled, including training the engineers***

Customer Experiences: Tau Generation2 Tools (UML-based)

- “We believe that the automatic implementation of visually defined software architectures is the next revolution in software development. Our overall evaluation of Tau/Developer was positive as the tool and the training exceeded our expectations.”
 - **Jason Smith, Raytheon**
- “At Nokia Research Center, we have evaluated Tau/Developer since early 2002, and we are impressed with the tool's strong visual modeling and simulation capabilities and its overall user-friendliness. As for the evolving UML 2.0, we view it as a very important technology platform, and we believe it will significantly increase productivity compared to traditional development approaches.”
 - **Ari Ahtiainen, Senior Research Manager, Nokia Research Center**

Wrap Up

- Software development must radically improve or face consequences
- Solution to software development paradox is not a silver bullet, but a language-tool-process transformation
- UML2-based development tools can realize potential of OMG's Model Driven Architecture
 - automate generation of production quality code
 - automate system validation & verification throughout software lifecycle
- Telelogic leads by example in UML 2.0 standardization and tool innovation
- Tau/Developer is an existence proof
 - shows how a mature architectural blueprint language (UML 2.0) can be synergistically combined with a power tool to automate code generation and verification

Resources

- Information about U2 Partners' proposals for UML 2.0
 - www.u2-partners.org
- Information about Tau Generation2 products
 - www.taug2.com