



What do building construction and software engineering have in common?

Larry Maccherone

Manager of Software Assurance Initiatives

CyLab - Carnegie Mellon



Creating secure software is like constructing a building to be secure

of course
you need good locks...
(authentication/authorization)



and blinds to keep out
prying eyes...
(encryption)



but the most critical
element is...
to start with strong materials
(defect-free components)

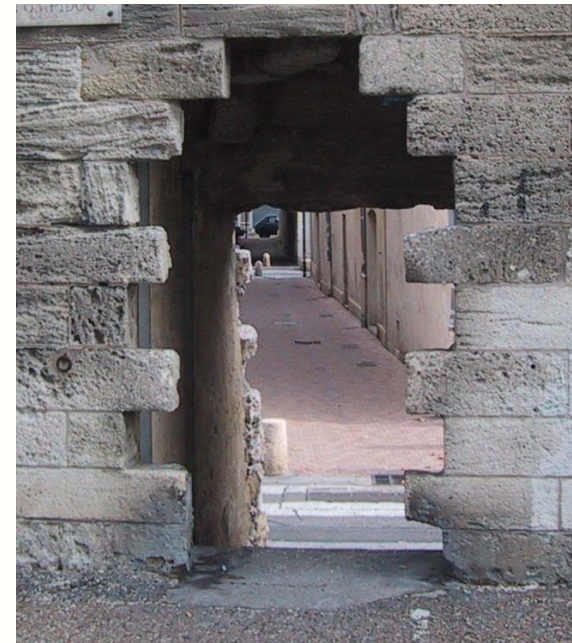
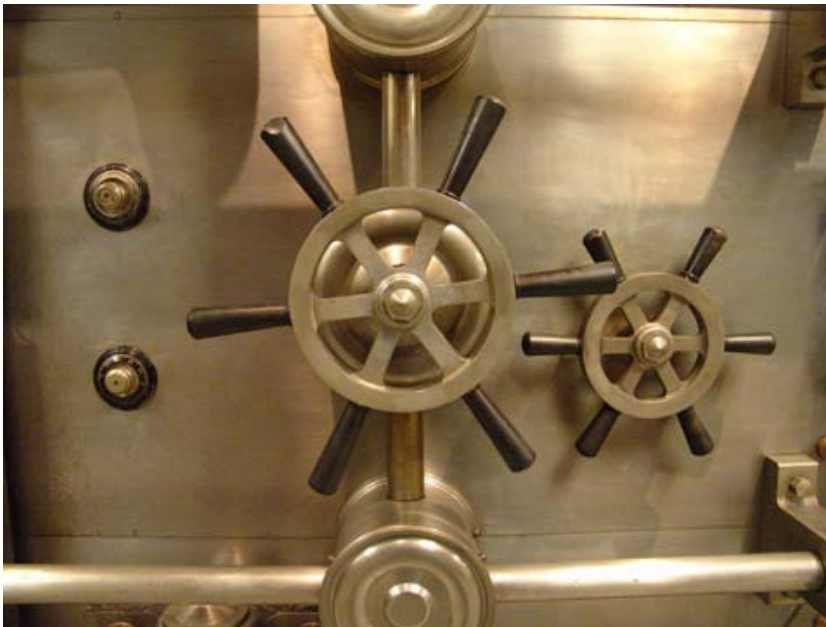


...and put them together
carefully (software engineering)

The bad guys...

don't break your authentication
or crack your encryption... that's
too hard

... and they generally don't need
to because ...
they can just bust through the
walls by exploiting defects



Software defects are exploited

Over 90% of cybersecurity incidents reported to CERT/CC can be traced to software defects

- SQL Injection
- URL Munging
- Cross-site scripting

... as well as

- Buffer overflows

... and many more



Security must be built-in...

Security is not...

- A list of features (encryption, authentication, etc.)
- Something that can be bolted on (firewalls, IDS, etc.)
- Magic fairy pixie dust sprinkled on by some consultant

Security must be built-in...

- By properly skilled/trained professionals with ongoing access to detailed information about the ever changing known set of vulnerable coding idioms (**people**);
- Using sound engineering practices and following a high maturity process (**process**); and
- Taking advantage of appropriate tools (**technology**);



...it cannot be bolted-on



Software Assurance

Software Assurance: confidence that software does what it is supposed to and **nothing more**

- **People:**
 - Training
 - Team formation and roles
 - Reference materials
 - Certifications
- **Process:**
 - Processes
 - Criteria like CMM, ISO-9000, OCTAVE (Risk)
 - Specific process instantiations - Team Software Process (TSP)
 - Design – Formal design and general design process
 - Process parts (e.g. Code review/inspection)
- **Acquisition/business/ROI:**
 - Standard set(s) of verifiable product characteristics
 - Product certification (beyond Common Criteria)
 - Business-level risk analysis
- **Technology:**
 - Language evolution
 - Climb the abstraction ladder – higher-level languages, aspects, executable and other modeling, verified frameworks and code generation
 - Protecting the programmer from himself – strong typing, “safe” subsets and library evolution
 - Resilient systems – fine grain architectures, isolation, virtual machines
 - Automated verification – static analysis, model/code agreement
 - Dynamic assurances – in testing and in use
 - Proof carrying code
 - Testing

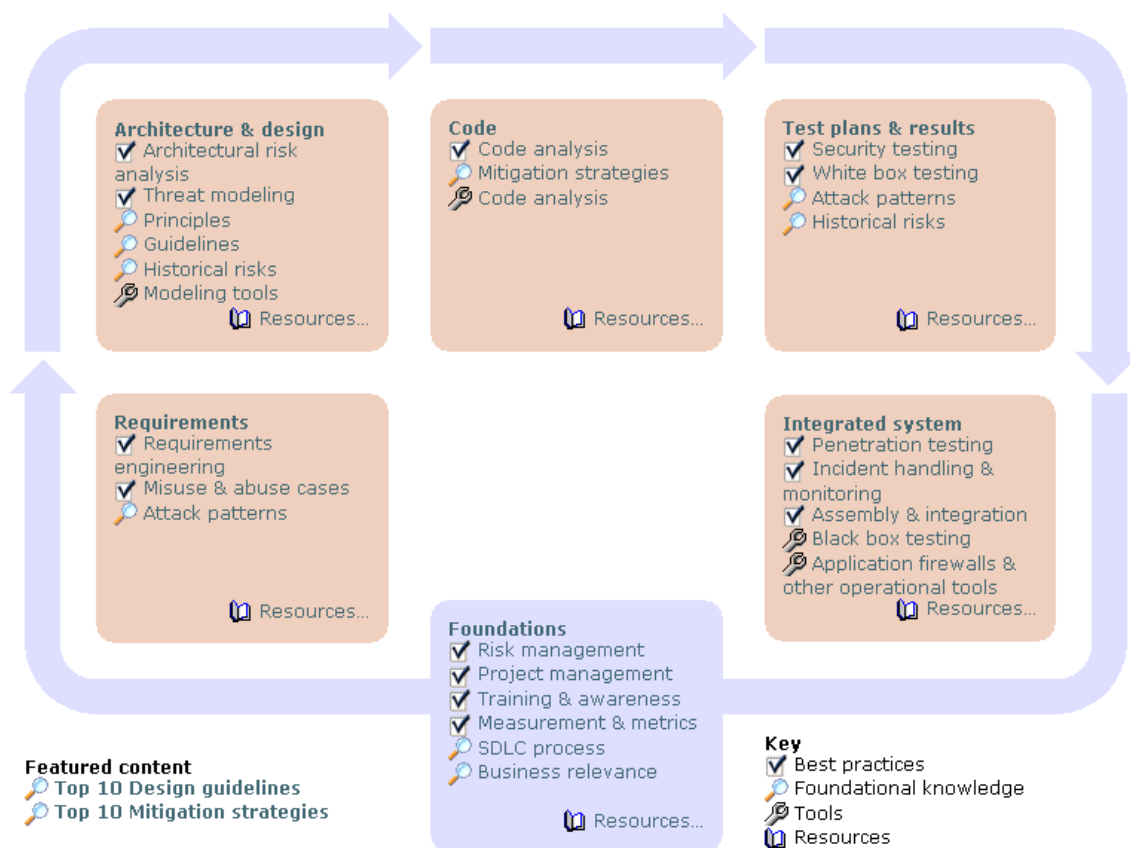


The Build-Security-In content catalog

- **Developer focus**
 - **Best practices** for each SDLC artifact
 - Requirements - Requirements engineering
 - Architecture and design - Architectural risk analysis
 - Code - Code analysis
 - Test plans and results - Security testing
 - Integrated system – Penetration testing
 - **Knowledge** to be referenced throughout the SDLC
 - Design guidelines
 - Mitigation strategies (detailed detection/elimination guidance for the most troublesome coding idioms)
 - Principles
 - Guidance on how to utilize **Tools**
- **Manager and executive focus**
 - Risk analysis
 - Project management
 - Measurement & metrics
 - SDLC process including the various CMMs
 - Business relevance



The Build-Security-In content catalog



Navigation alternatives

- Lifecycle touchpoints view
- Top 10 lists
- Filtered tree (by technology: C/C++, Java, Python, .NET, Windows, Linux, etc.; role: Architect, Developer, Manager, etc.; etc.)
- Favorites

Other elements:

- News →
- Discussion →
- Articles, more... →



The Build-Security-In content catalog

- **Expert team**

- Gary McGraw
 - CTO – Cigital and security expert
 - Co-author of *Exploiting Software*
 - Co-author of *Building Secure Software*
 - Co-author of *Securing Java*
- Ken VanWyk
 - Security expert
 - Author of *Secure Coding: Principles and Practice*
- Robert Seacord and Dan Plakosh
 - Technical experts from SEI and CERT/CC
 - Authors of forthcoming book in the SEI series on Software Engineering, *Secure Coding in C and C++*
- 15 experts from CyLab, SEI, CERT/CC and Cigital

- **Industry/Government Participation (technical and executive)**

- | | | | | |
|-------------|----------|-------|-------------|---|
| – Microsoft | – IBM | – DHS | – NIST | → |
| – HP | – SAP | – DOD | – FAA | → |
| – Cisco | – Oracle | – NSA | – Others... | → |





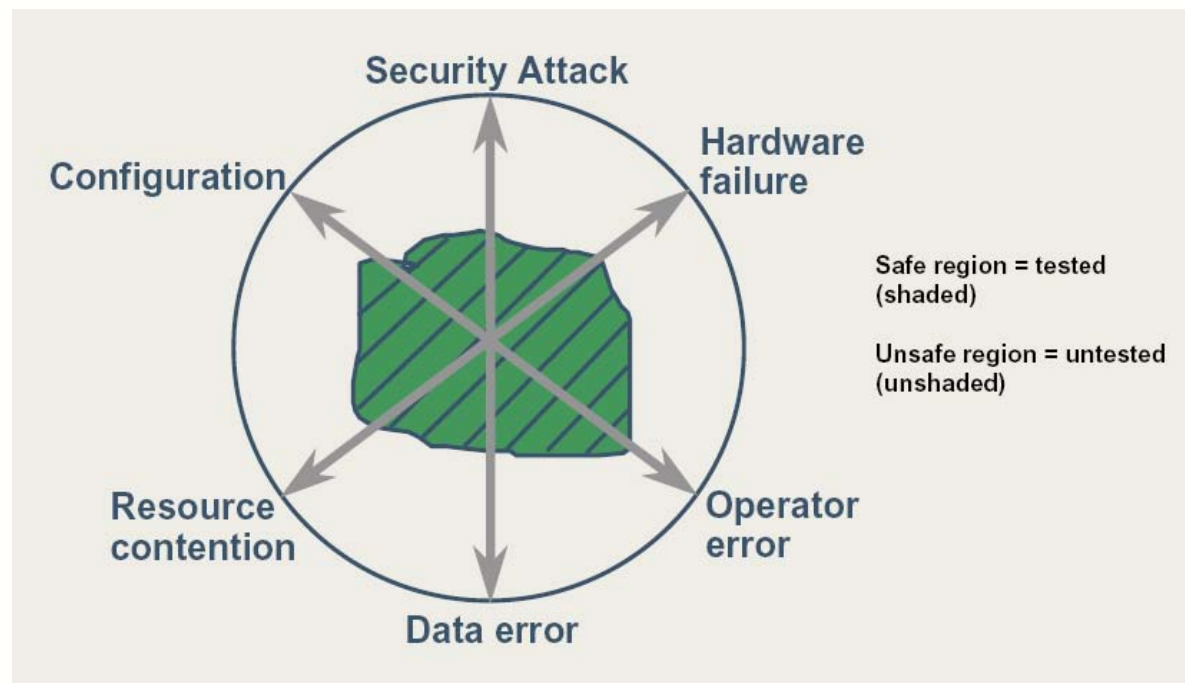
Team Software Process (TSP) and TSP Secure



- SEI – Watts Humphrey
- Covers most of CMM Level 5
- Highly effective (Data below from 20 recent real-world projects)
 - 60 defects per million lines of code (5.2 Sigma)
(typical: 1000-5000)
 - Minus 20% to plus 27% schedule deviation
(typical: 58% of projects are more than 100% late)
 - 78% Improvement in productivity
- Fast implementation – 18 months typical (CMM level 5=5 years)
- Platform for sustaining improvements
 - Takes developer psychology into account
 - Continuous improvement loops
- TSP Secure
 - Specific practices for security to plug into TSP (or other processes) →
 - Deep, hands on training – find 30 defects in a web app →

TSP – The problem with testing

- Only covers a small percentage of possible executions (typically 0.1% to 5.0%)
- Stays within constraints and assumptions



... instead, we must have virtually defect-free software before releasing to test



TSP – Lessons of Deming

The continuous improvement loop

1. Say what you do (process documented including all the “rules” and stated intent of no defects going into test)
2. Do what you say (micro-training, see 4)
3. Record what you did (easy → source code)
4. Measure the difference (verification)
5. Act on the difference (train, modify rules)

Software is the only modern technology that ignores quality until test.



TSP – Principles are not enough

Some software engineering principles

- The need for accurate plans
- The importance of detailed, verifiable designs
- Early defect removal
- Effective inspections done often
- Focus on quality throughout

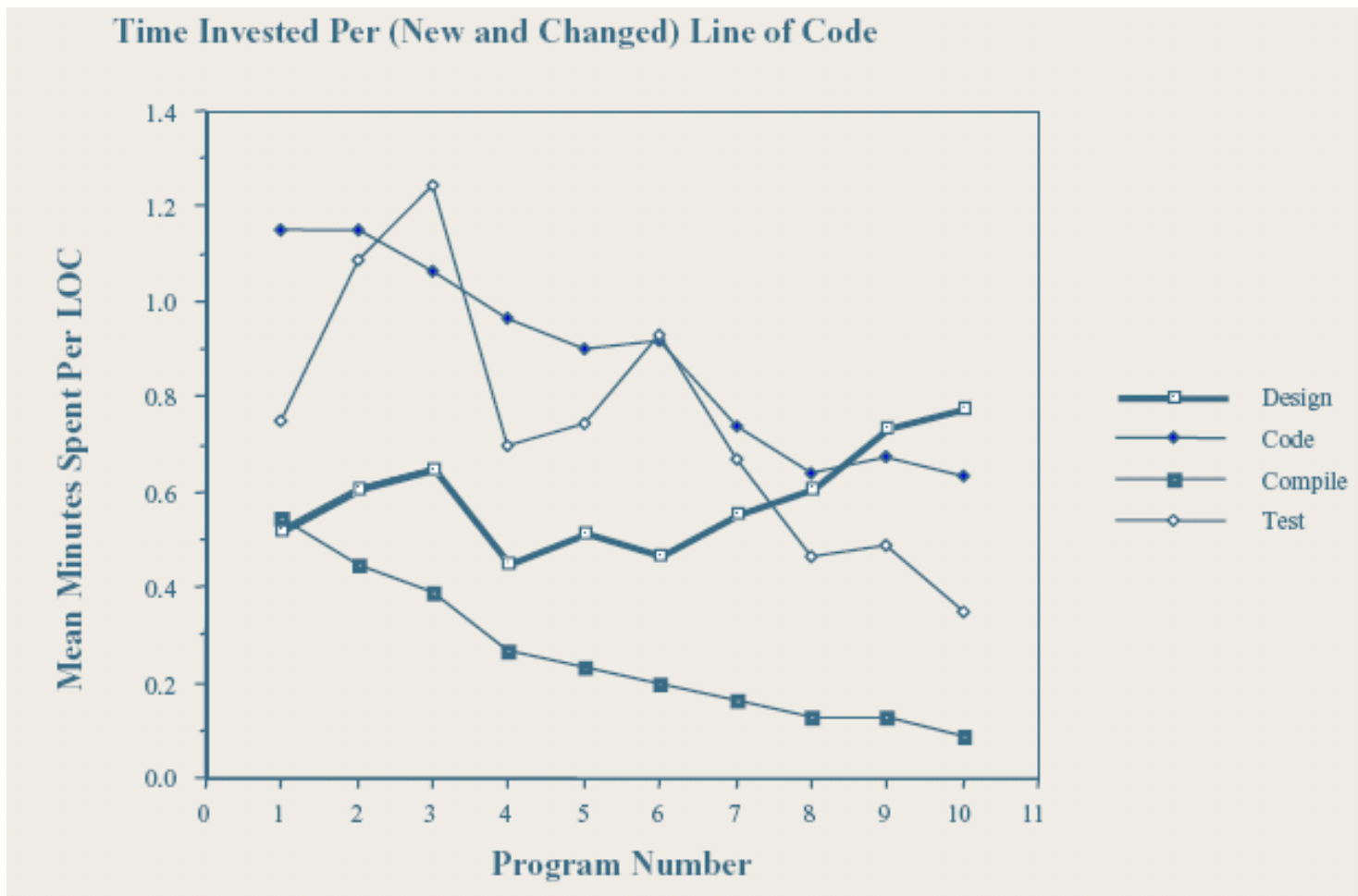
Why are these principles not applied?

To apply, you need:

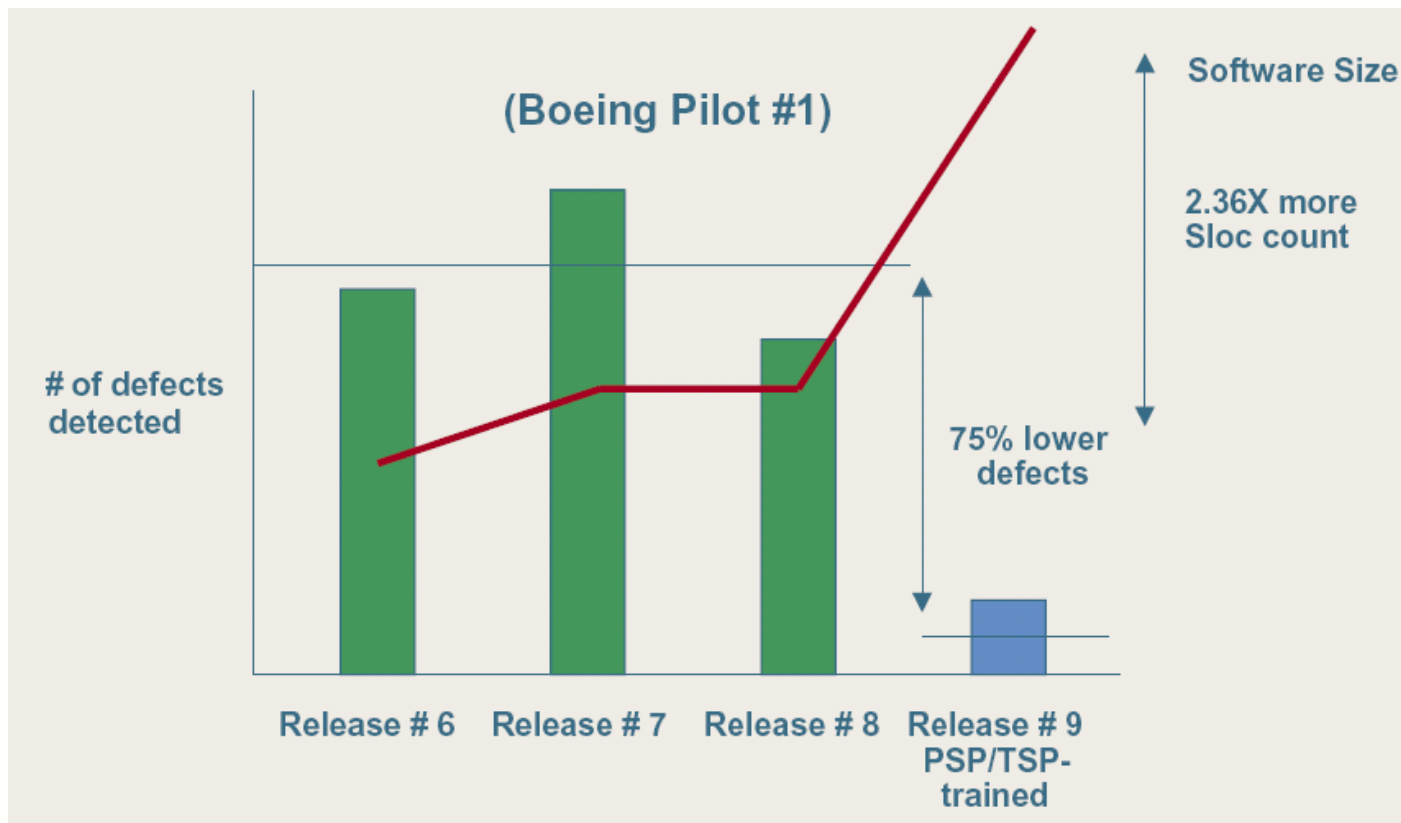
- A supportive infrastructure and environment
- An operational process (rules and steps) to put the principles into practice
- A measurement system to manage and control the results
- Conviction from the developers in the value of these principles

...TSP provides these

TSP – PSP gains conviction



TSP – TSP provides the rest



- Over 10x improvement in defect density
- System test went from 30 days to 4





Research example – Fluid

Engineering properties:

- **Safe concurrency**
 - Race conditions
 - Lock management
 - Single thread concurrency control
 - Lock ordering and deadlocks
- **Code safety**
 - Ignored exceptions
 - Appropriate typing
- **Policy compliance**
 - API policy compliance
 - Framework compliance
 - Object references and aliasing
 - Patterns, uses, structure
- **Real time**
 - Real-time thread/memory policies

Recent successful engagement:

- Reviewed tier 1 J2EE Application Server – 350,000 +/- lines of code
- Focused on concurrency – very hard to inspect/test
- Assured thousands of correct locks, found 120+ problems
- Re-factored key clustering code for increased scalability, performance, and reliability and proposed other re-factorings – **surprise outcome**
- Reverse engineered design intent of purchased and/or “mangled” code – **#1 priority for engagement**
- Strong desire for tools to become a part of regular development process





Research example – Fluid

Quotes from users:

- “So this tool will let me put in the design intent and then tells me if it is consistent with the code?”
- “To me the most valuable thing is the basic fact that you’ve given us a methodology to document the concurrency related design intent.
- “I’m actually considering implementing a policy that you can’t add a ‘synchronize’ to the code without documenting [in Fluid notation] what region it applies to.”

Tool/technology development principles:

- Positive assurance as opposed to bug hunting
- Usability
 - Familiar environment (Eclipse)
 - The model is in the code and the code is the ONLY model that matters

Fluid what’s next:

- Spin-off technology to tool company
- Integration with commercial products →
- More engagements with CyLab members for new capabilities →



CyLab

- Research
 - Over \$12M/year projected starting 2007
- Education
 - Information Networking Institute (INI) – 15 years
 - Curriculum building program
- Working Groups
 - Software Assurance
 - provides input/content to Build-Security-In catalog
 - Workshops and other events
 - Education/Curriculum
 - Standard for static analysis tools
 - Risk Management
 - Privacy and Policy
- International
 - CyLab Korea
 - CyLab Japan
 - CyLab Greece
 - India
 - Middle East
 - Others...





Contact

Larry Maccherone

Manager of Software Assurance Initiatives

CyLab – Carnegie Mellon

LMaccherone@cmu.edu

