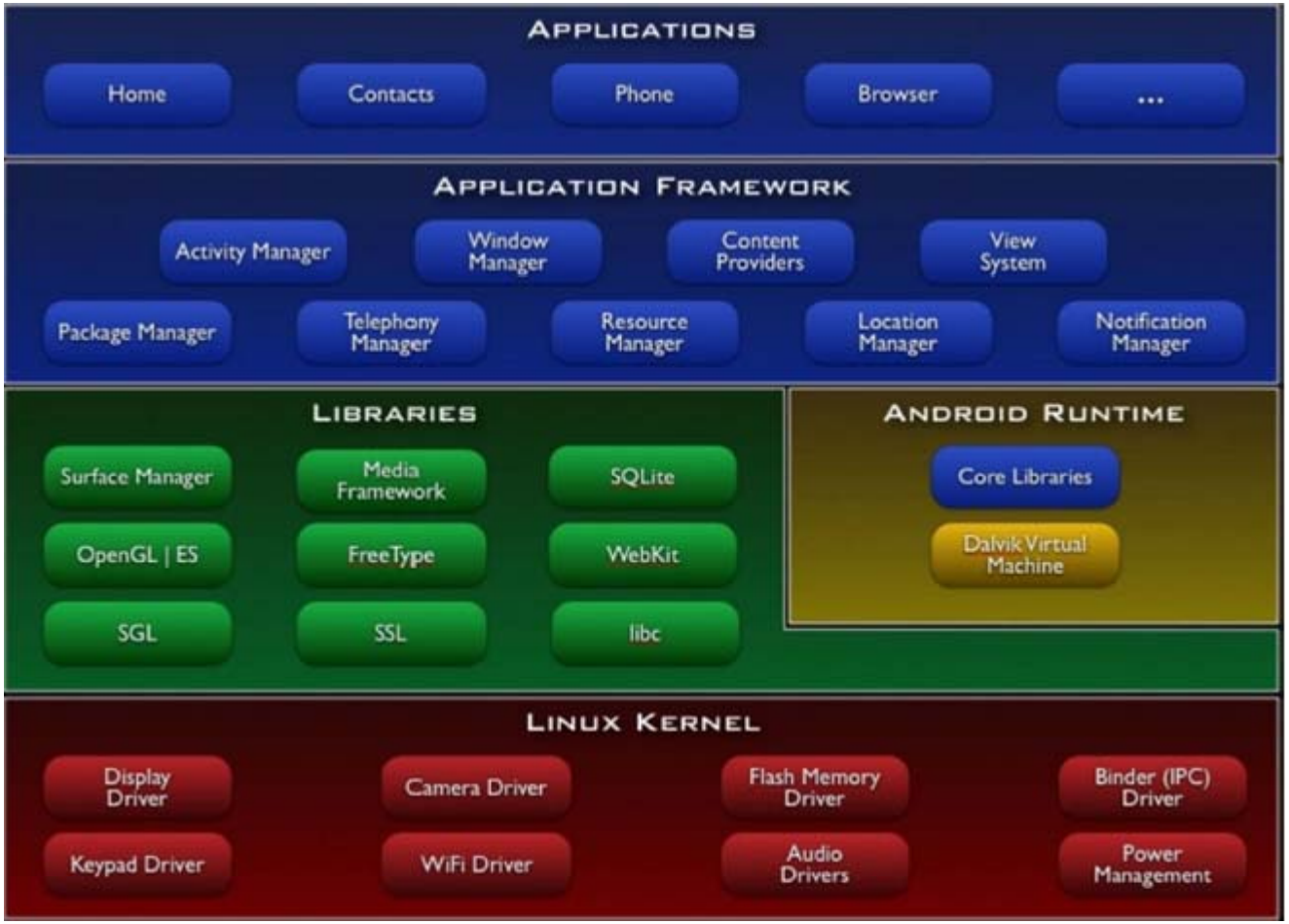
CPSC 875

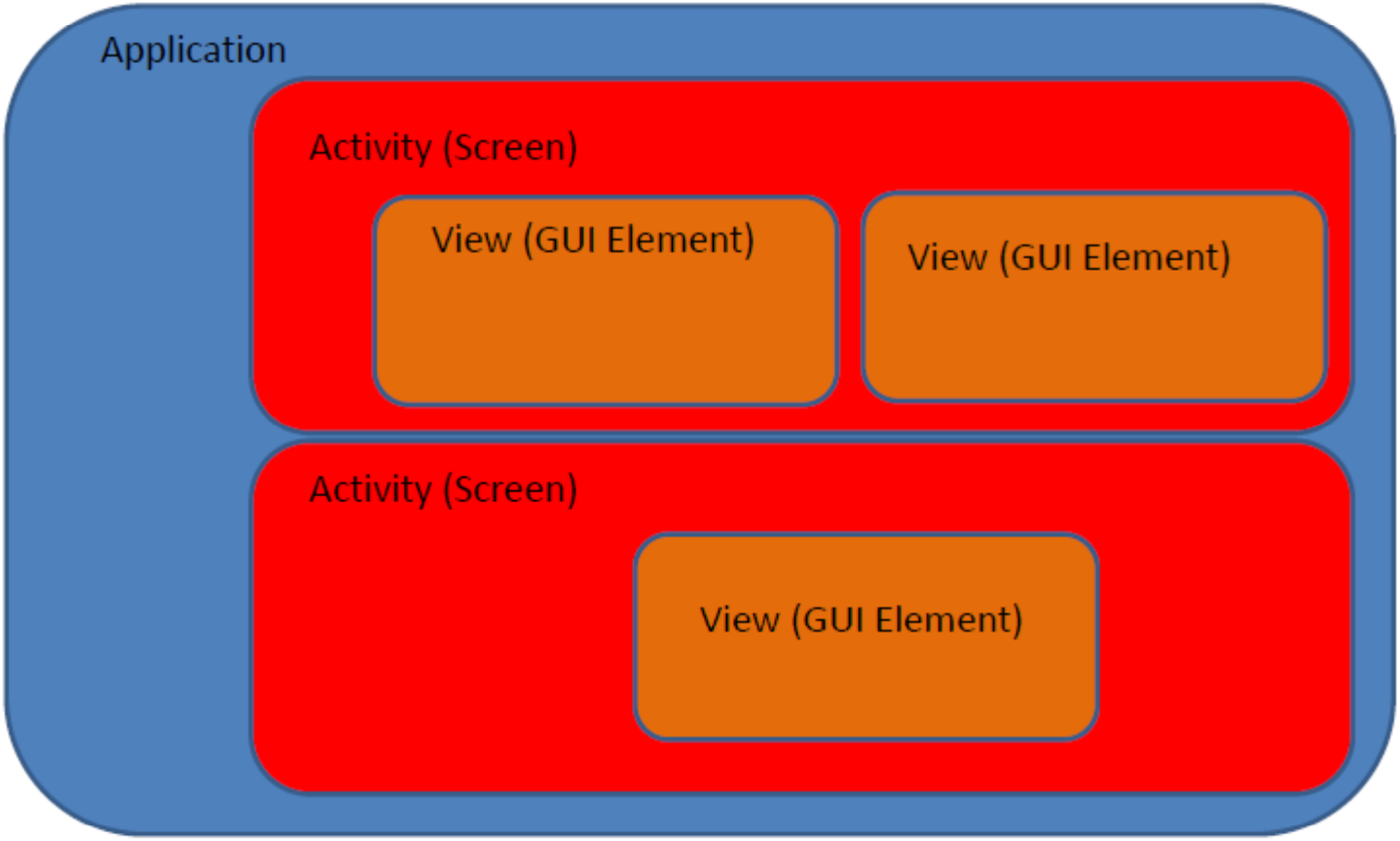
John D. McGregor

C20 – Technical Debt

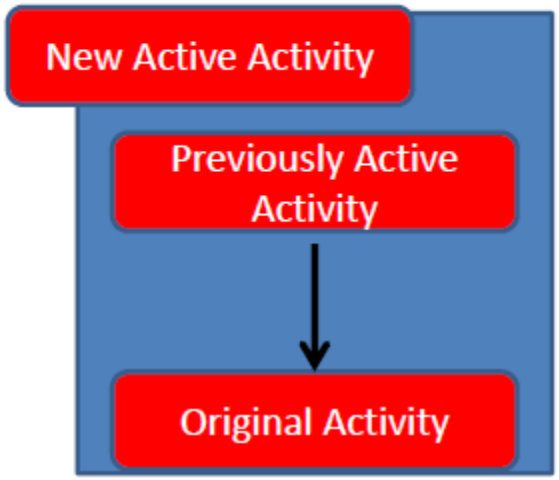
Android architecture



App architecture



Activities



Stack top (newest)

(oldest)

Activity States

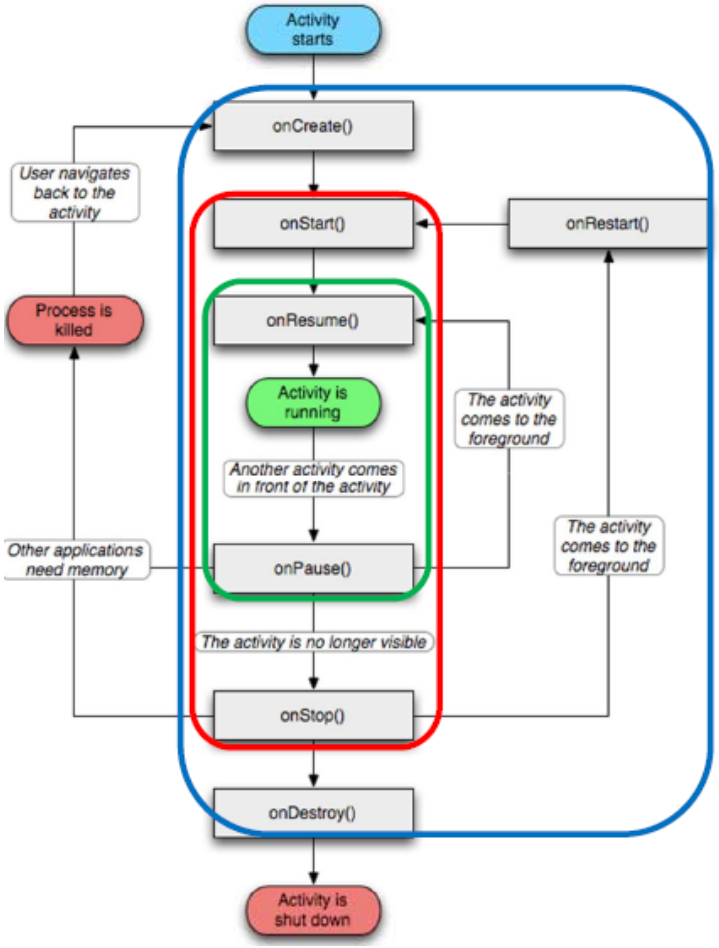
Active
Foreground – receiving input

Paused
Visible but obscured

Stopped
No longer visible, still in memory

Inactive
Not visible, not in memory (terminated)

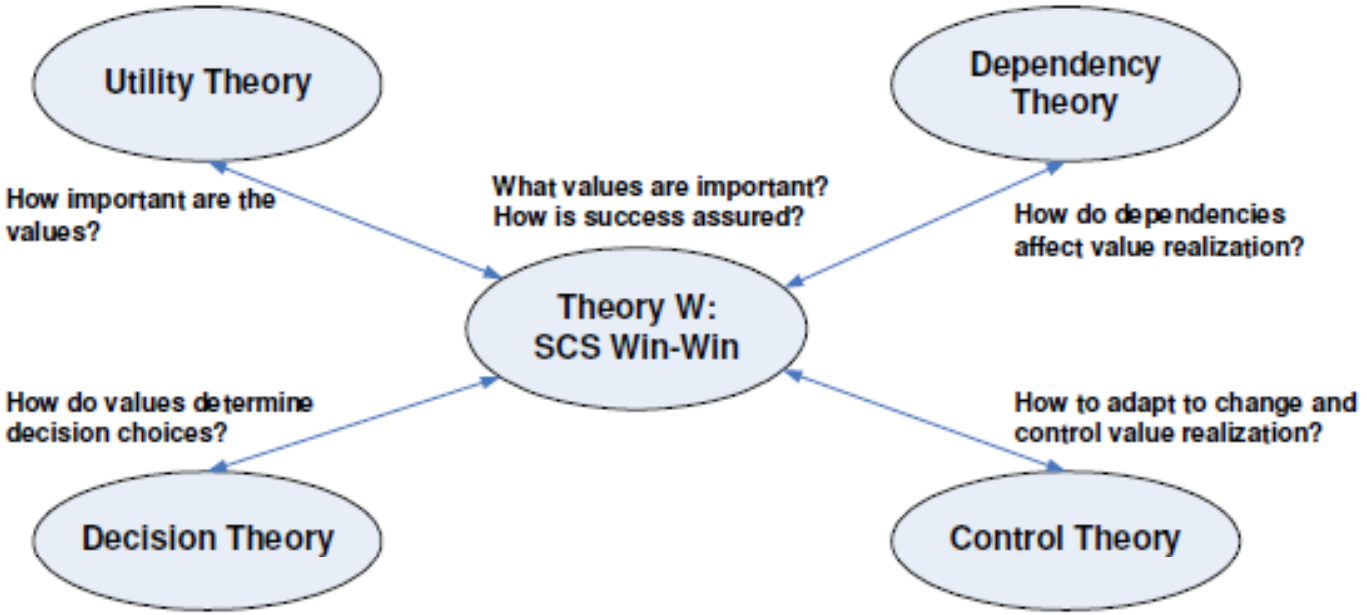
Activity states



Model View ViewModel

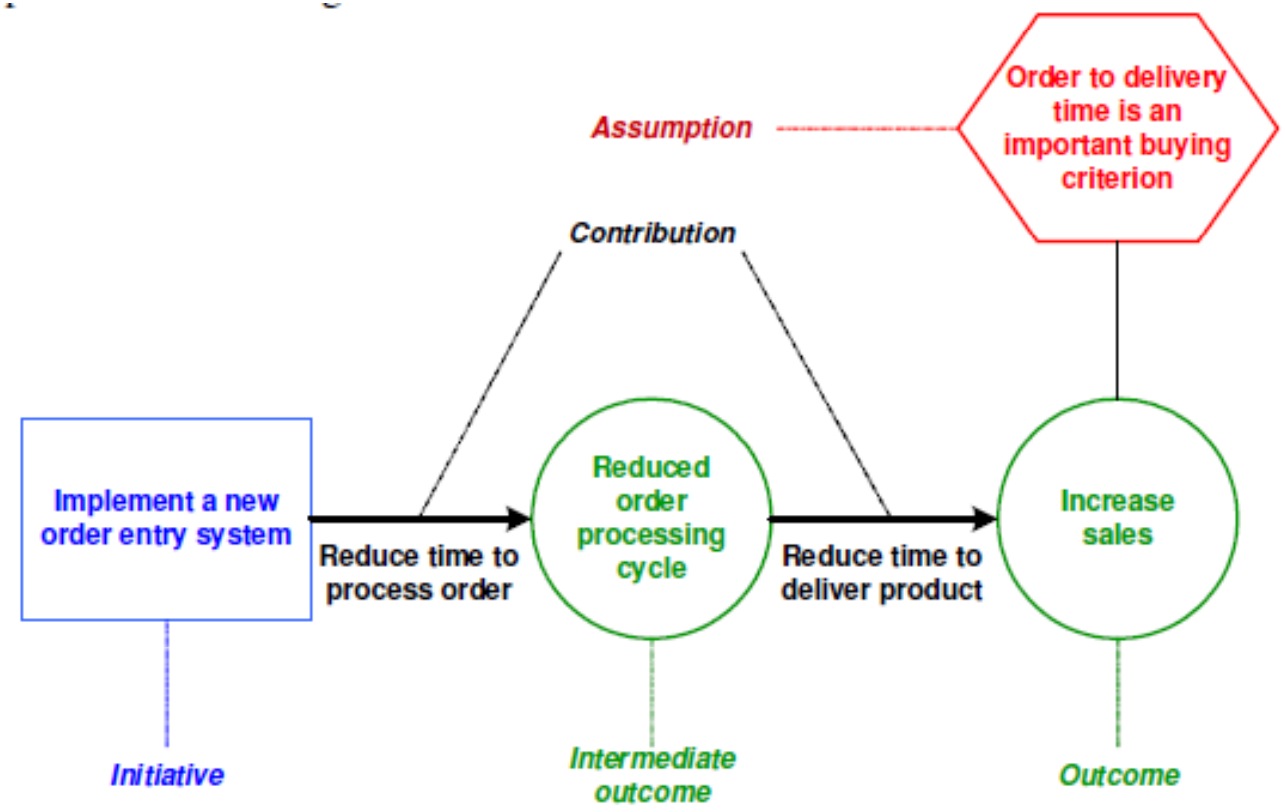
- **What is MVVM ?**
- An architectural separation pattern
- A variant of the Model-View-Controller that leverages Data Binding
- **Parts of the MVVM Pattern**
- **View**
- User Interface
- **View Model**
- Presentation Logic
- State
- **Model**
- Data
- Business Logic

Value-based SE

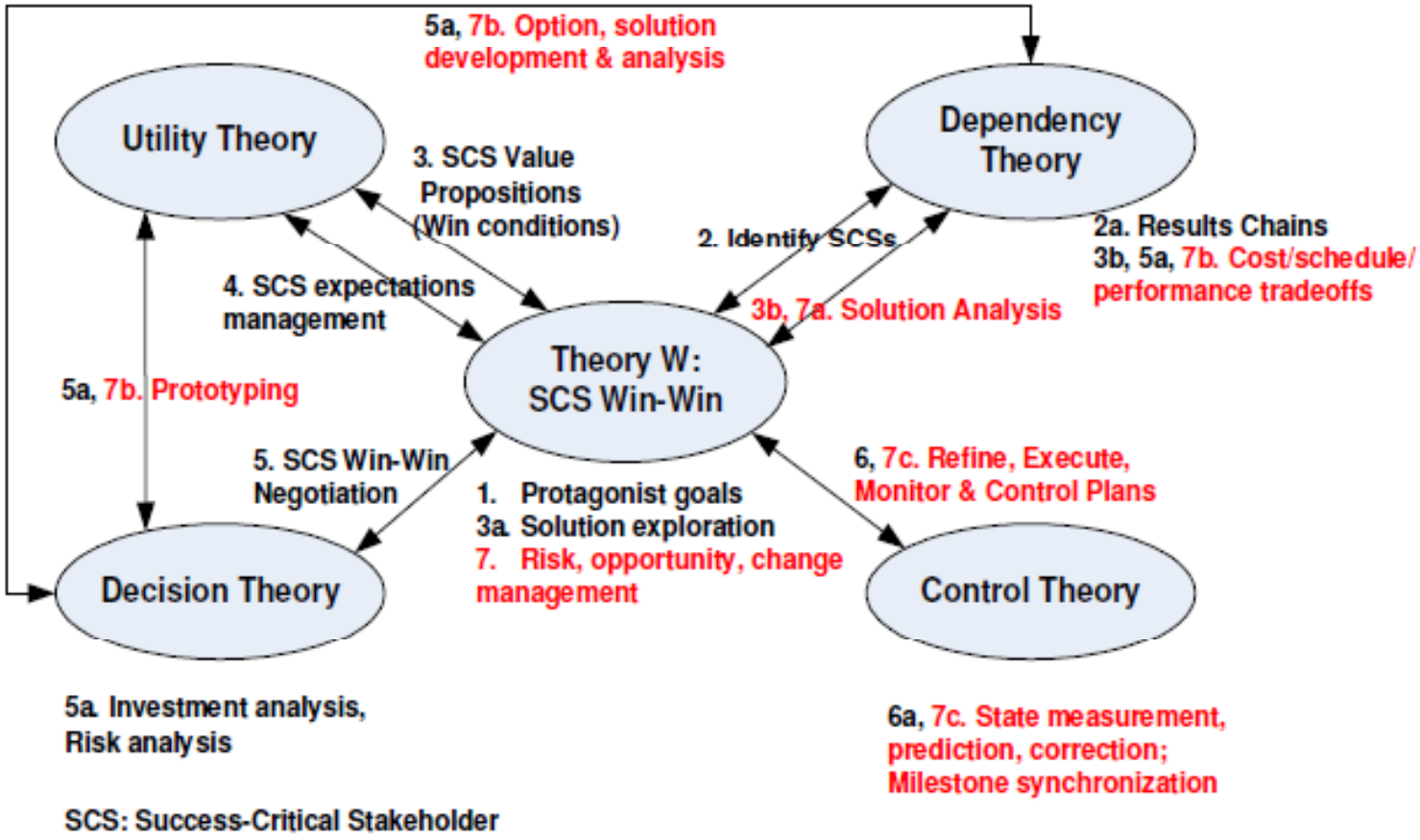


SCS – success-critical stakeholder

A process


Order to delivery time is an important buying criterion
Assumption
Contribution
Implement a new order entry system
Reduce time to process order
Reduced order processing cycle
Reduce time to deliver product
Increase sales
Initiative
Intermediate outcome
Outcome

Value-based result



- We don’t have time to upgrade our compiler version for now, we’ll do it later.
- We should probably use this design but it will takes 6 months compare to this reasonable one which will take 2 months.
- We’re reaching some crash after a 100 hours stress run. No customer will go that far, let’s ship it for now !
- Our code doesn’t comply with industry standard. Let’s ship it anyway and will fix it later.
- http://www.fredberinger.com/avoid-bankruptcy-manage-your-technical-debt/
- if it actually took them 5 days but they think it would have taken them 3 days with a clean system, then you paid 2 days of effort as interest on your technical debt

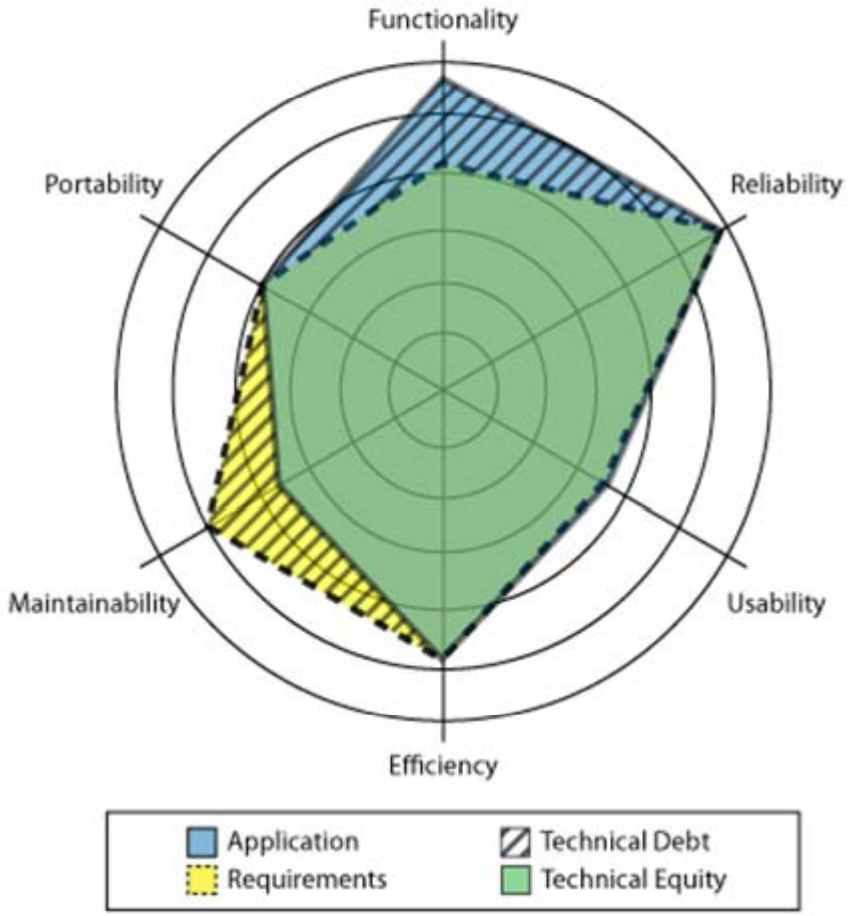
Technical debt

- Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite… The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation, object-oriented or otherwise.

Technical debt - 2

- Ongoing development in the upstream project can increase the cost of “paying off the debt” in the future. A team should take opportunities, on a regular basis, to pay back (or pay off) this debt. Either reserve a percentage of your development cycle or dedicate an entire cycle to complete this work. If you don’t, it **will** come back to haunt you. If your kludge of a solution doesn’t come back to bite the development team, it will probably haunt the help desk, support team, or someone else downstream.
- http://thecriticalpath.info/2011/01/16/technical-debt/

Technical debt is a gap



http://blog.acrowire.com/technical-debt/the-stakeholder-perspective-conclusion/

The metaphor

- To date, technical debt has been used as a metaphor and rhetorical device within the agile community with increasingly recognized utility for technical communication and for communication between engineers and executives. The technical debt concept is gaining traction as a way to focus on the long-term management of accidental complexities created by short-term compromises.
- http://www.sei.cmu.edu/community/td2011/upload/foser076-brown.pdf

Why debt?

- There is an optimization problem where optimizing for the short-term puts the long-term into economic and technical jeopardy when debt is unmanaged.
- Design short-cuts can give the perception of success until their consequences start slowing projects down.
- Development decisions, especially architectural ones, need to be actively managed and continuously analyzed quantitatively as they incur cost, value, and debt.

Debt, costs, and value

- You are allowed to run up debt until the lender questions your ability to repay
- Periodically, you need to pay off some debt in order to be able to borrow again when you need to.
- That means planning for time in the development process to pay back.

Properties of debt

- *Visibility. Significant problems arise when debt is not visible.* In many cases, it is (or was) known to some people but it is not visible enough to others who eventually have to pay for it.
- *Value. In its financial use, debt when managed correctly is a* device to create value
- *Present value. In addition to the overall potential system* value enabled by technical debt, the present value of the costs incurred as a result of the debt, including the time-to impact and uncertainty of impact, must be mapped to the overall cost-benefit analysis.

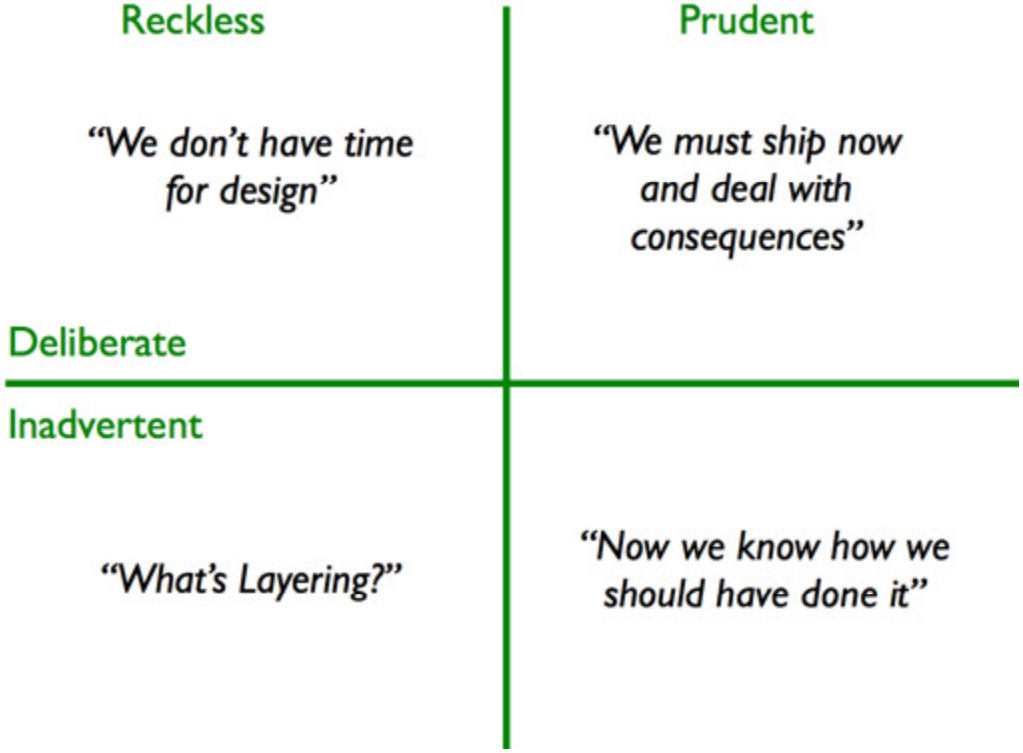
Properties of debt - 2

- *Debt accretion. Debt does not necessarily combine* additively, but super-additively in the sense that taking on too much debt leads a system into a bad, perhaps irreparable state
- *Environment. In software engineering projects, debt is* relative to a given or assumed environment.
- *Origin of debt. It is important to distinguish sharply between* strategic debt, taken on for some advantage, and unintentional debt, that is taken on either through poor practices or simply because the environment changed in a way that created a mismatch that reduces system value.

Properties of debt - 3

- *Impact of debt. The locality (or lack thereof) of debt is* important: are the elements that need to be changed to repay a debt localized or widely scattered?

Classes of debt



Assessing Technical Debt

- **Complexity**
- **Code Duplication**
- **Documentation Debt**
- **Testing Debt**
- **Architectural Debt**

Research issues

- **Refactoring**
- **Architectural issues**
- **Identifying dominant sources of debt**
- **Measurement issues**
- **Non-functional artifacts**
- **Monitoring**
- **Process issues**

Entropy reduction

- *Use code tags.*
- *Establish a rhythm.*
- *Time-box the ER activity*
- *Don’t ship the result*
- *Choose your language carefully*
- *Use ER to reinforce other values you deem important*
- *Don’t commit unless you can deliver.*

Why no-one repays technical debt

- **Business people don’t see technical debt.**
- **Perceived low ROI.**
- **Developers don’t like repaying technical debts.**
- **Development processes don’t focus on it.**

- http://innovationgames.com/wp-content/uploads/2010/04/entropy-reduction.pdf