1.
SWT bedeutet Software Engineering: Das ist das systematische, quantifizierbare Vorgehen, um Software zu entwickeln, einzusetzen und zu warten (zu pflegen). Software Engineering befasst sich mit dem Ermitteln der Ausgangssituation und Anforderungen, dem Entwurfsmuster, der Entwicklung von Softwaresystemen und Schnittstellen zwischen verschiedenen Softwaresystemen, der Inbetriebnahme, der Wartung und umfasst daher viele Bereiche: Analyse (Analyse der Ist-Situation), Definition (Anforderungsanalyse, Produktdefinition), Konzeption/Entwurf (Bestimmen des Sollkonzepts, Produktentwurf, Mock-Up, Prototyping, Entwurfsmodelle), Implementierung (Modellierung, Programmierung, Customizing), Einführung (Installation, Deploying, Anwender-Training), Wartung-/Pflege (Erweiterungen, Customizing, Fehlerbehebung, Anwender-Support).
SWT nutzt Methoden, Werkzeuge, Maßsysteme Standards und Erfahrung, um Software systematisch und zweckorientiert (zielgerichtet) zu entwickeln.

- Bei den Methoden haben sich in der SW-Entwicklung als best practices unter anderem die Prinzipien Prinzip der Abstraktion
- Prinzip der Strukturierung
- Prinzip der Modelarisierung
- Prinzip der Hierarchisierung
- Prinzip der Standardisierung

bewährt.

2.
Gute Software erfüllt die gestellten Anforderungen.
Nach ISO 9216 soll Software

1. funktionell,
2. kompatibel (Interoperabilität zwischen verschiedenen SW-Systemen)
3. sicher (vertraulich (information hiding), integrativ)
4. effizient
5. zuverlässig (fehlertolerant)
6. benutzbar
7. wartbar (analysierbar, anpassbar)
8. protierbar

sein.
Zu guter Softwarequalität gehört nach meinem Verständnis nicht nur die Software an sich, sondern auch ein gute technische Dokumentation für die Anwender und code-Kommentare für die Softwareentwickler. Dokumentation wird leider immer etwas vernachlässigt, da diese einen gewissen Aufwand bedeutet und keinen ausführbaren Code generiert 😉. Und dennoch ist diese so wichtig (Das erfahre ich aktuell in meinem Berufsalltag.).
Gute Softwarequalität ermöglicht auch einfachere Festlegung von Standards. Schnittstellen zwischen verschiedenen Software-Systemen sind einfacher entwickeln und pflegen. Gute SW ist fehlertolerant und robust. Das meint, dass falsche Benutzer-Eingaben mit passender Fehlermeldung angefangen werden, dass auf Typenverträglichkeit geprüft wird, Stack overflow und Buffer overflow, Code Injection abgefangen werden. Gute SW-Qualität ermöglicht eine besseres co-working von verschiedenen Entwicklern. Installationen, Roll-outs, deploying, und Updates lassen sich natürlich mit guter Software einfacher und effizienter realisieren.

3.
Gute Softwarequalität ist zunächst die Umsetzung der Anforderungen bzw. der Erwartungen, die an die Software gestellt werden.
Gute Software unterstützt den Anwender in der Nutzung und ist gut zu bedienen. Dazu gehören eine übersichtliche GUI nach den Regeln der Usability in der MCK, ein gutes Hilfesystem, erwartbare Reaktionen auf Eingaben, gute Fehlerbehandlung.
Zu guter Software gehört auch leichte Erlernbarkeit/intuitive Benutzung durch die Anwender (jeder Software-Entwickler sollte daran denken, dass Software für den Benutzer entwickelt wird.
Softwarequalität ist aber auch effizienter Code mit aussagefähiger Zeilendokumentation. Gute Software löst die Anforderungen auch bei großer Komplexität.
Komplexe Softwaresystemen sollten modular strukturiert (Prinzip der Modularisierung) sein
Und vor allem muss Software zweckorientiert sein (Stichwort Anforderungen), d. h. sie muss das leisten, wofür sie gedacht ist.

4.
Eines der größten Probleme in Softwareprojekten ist das Nichterreichen des Projektziels. Hierzu gibt es in der Geschichte zahlreiche Beispiele zu (oft sehr kostspieligen) Softwareprojekten, bei denen das Projektziel nicht erreicht wurde. Oft erfüllt die Software nicht die Erwartungen oder das Projektziel wird überhaupt nicht erreicht. Hierfür gibt es verschiedenen Ursachen. Eine wesentliche Ursache für das Nichterreichen der Projektziels ist eine unvollständige bzw. ungenügende Analyse der Ist-Situation und Anforderungs-Analyse. Das erlebe ich fast täglich in meinem beruflichen Alltag. Die Hauptgründe der mangelhaften Anforderungs-Analyse sind Zeit-Kostendruck (meist durch das Management ausgeübt), mangelnde Qualifikation bzw. Kenntnisse der Beteiligten sowohl auf der Seite des Auftraggebers als auch beim Auftragnehmer und zu geringe Gewichtung der Priorität der Analyse- und Definitions-Phase. Unzureichende Sorgfalt in den Analysephasen führen im weiteren Projektverlauf zur Überschreitung des Zeit- und Finanz-Budgets zu Nichterreichung oder nur teilweisen Erreichung des Projektziels oder schlimmstenfalls gar zum Projektabbruch (Das habe ich alles schon erlebt.). Problematisch ist auch, wenn die Ansprechpartner beim Kunden nicht über das erforderliche Wissen bzw. über die notwendigen Kenntnisse verfügen. Oft kann der Kunde seine Anforderungen nicht präzise formulieren. Hierzu bedarf es beim Auftragnehmer geschulte Mitarbeiter, welche die Interviews mit dem Kunden richtig vorbereiten und führen.
Mangelnde Projektierung ist ein weiterer Grund für das Nichterreichen des Projektziels.
Die Überschreitung des Zeit- und Kostenplans folgt dem Nichterreichen des Projektziels. Durch verschiedene Gründe, wie die bereits Genannten, fehlende bzw. unzureichende Beteiligung des Managements, der IT-Entscheider und der Anwender, fehlende fachliche Kenntnisse, Anwendung falscher Methoden und Werkzeuge, mangelhafte Analyse der Ist-Situation, fehlerhafte Annahmen von Voraussetzungen und Bedingungen, zu viele nachträgliche Änderungen werden in den meisten Softwareprojekten der Zeit- und Kostenrahmen nicht eingehalten. Als IT-Entscheider auf der Kundenseite würde ich in großen Projekten einen externen Dienstleister mit dem Projektmanagement beauftragen und niemals das Unternehmen, dass das Softwaresystem entwickelt.
Das dritte große Problem in Softwareprojekten ist der Abbruch dieser auch aus den bereits genannten Gründen. (eines der missglücktes Softwareprojekte ist z. B. die gescheiterte Umstellung des WaWi-System auf SAP beim Discounter Lidl: 500-Millionen-Euro-Projekt scheitert <https://t3n.de/news/500-millionen-euro-projekt-scheitert-lidl-blaest-sap-software-ab-1095673%3e>)
Für alle der genannten Probleme ist die Unterschätzung der inhärenten Projektkomplexität oder durch menschliche Interaktion herbeigeführte Komplexität eine der häufigsten Ursachen.

5.
Die Phasen des Softwarelebenszyklus sind:

Phase	Ergebnis
Analysephase	Situationsstudie
Definitionsphase	Produkt-Definition
Entwurfsphase	Produkt-Entwurf
Implementierungsphase	Programme
Abnahme-/Einführungsphase	Installiertes Produkt
Wartungs-/Pflegephase	Gewartetes Produkt

6.
Ich habe eine Ausbildung zum Fachinformatiker für Anwendungsentwicklung und in meiner Tätigkeit als ERP- und CRM-Projektmitarbeiter in nahezu allen Phasen des Softwarelebenszyklus gearbeitet.
Am Anfang meiner beruflichen Karriere habe ich in der Analysephase die Aufbau-Organisation, realen Geschäftsprozesse (Ablauforganisation), die bestehende IT-Systemlandschaft, Benutzerberechtigungen, Schnittstellen zwischen dem abzulösenden Software-System und anderen Systemen etc. analysiert.
In der Definitionsphase wurden gemeinsam mit dem die Anforderungen an das neue System ermittelt und analysiert. In dieser Phase war ich an der Analyse und an der Ausarbeitung des Pflichtenhefts beteiligt. Später habe ich in der Entwurfsphase an der Daten-Modellierung, am Prototyping und Mock-Up mitgearbeitet. Die Modellierung erfolgte u. a. mit Hilfe der Werkzeuge UML-Klassendiagramm, use-case-Diagramm, ER-Diagramm. In der Implementierungsphase habe ich das System customized, Reports mit Hilfe von VB Script und MS SQL erstellt und AddOns entwickelt.
Am Ende der Projekt habe ich am deploying und in der Systemintegration mitgearbeitet und war auch mit der Anwender-Schulung und dem Support beauftragt.
Die Projekte, an denen ich beteiligt war, bezogen sich auf die Einrichtung, Anpassung und Erweiterung von ERP- und CRM-Standard-Software wie
Corporate WINLine (ERP-System)
CRM Plus (CRM-System)
Orcale CRM On Demand (CRM-System).

Prinzipiell arbeite ich gern in allen Phasen. Die Software-Entwicklung und -Anpassung (Implementierungsphase) und die Definitionsphase wecken aber mein besonderes Interesse.

Ich finde folgende Literatur für den Einstieg in das Software-Engineering sehr hilfreich:
SOFTWARE ENGINEERING - KOMPAKT<https://www.hanser-elibrary.com/doi/book/10.3139/9783446463653> METZNER Anja, Carl Hanser Verlag GmbH & Co. KG, 2020 DOI: 978-3-446-46365-3
Einführung in die Softwaretechnik <https://link.springer.com/book/10.1007/978-3-662-50263-1> BROY, Manfred, Springer-Verlag, Berlin Heidelberg, 2021, DOI: 10.1007/978-3-662-50263-1

Darüber hinaus interessiere ich mich für agiles Projektmanagement und habe in der Recherche das Buch
Agiles Projektmanagement im Berufsalltag <https://link.springer.com/book/10.1007/978-3-662-62810-2> ; KUSAY-MERKLE, Ursula, Springer-Gabler-Verlag, 2021, 2. Auflage, DOI: 10.1007/978-3-662-56800-2
als für mich zum Einstieg passend gefunden.

Last but not least erhoffe ich mir in diesem Modul Praxis im Umgang mit Git Hub <https://github.com> zu bekommen.